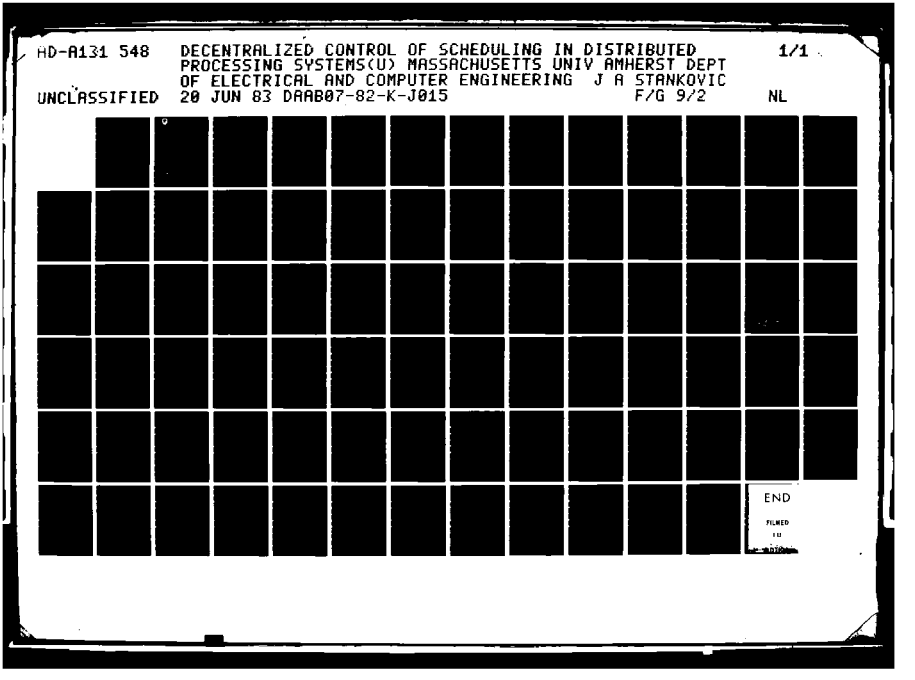
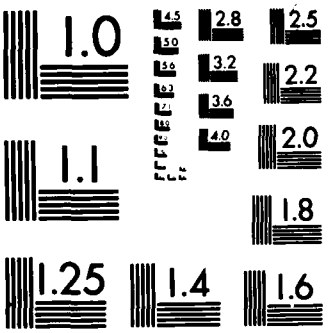

AD-A131 548 DECENTRALIZED CONTROL OF SCHEDULING IN DISTRIBUTED PROCESSING SYSTEMS(U) MASSACHUSETTS UNIV AMHERST DEPT OF ELECTRICAL AND COMPUTER ENGINEERING J A STANKOVIC
1/1
UNCLASSIFIED 20 JUN 83 DAAB07-82-K-J015 F/G 9/2 NL
END
FILMED



RESEARCH AND DEVELOPMENT TECHNICAL REPORT

CECOM

DECENTRALIZED CONTROL OF SCHEDULING IN DISTRIBUTED SYSTEMS

JOHN A. STANKOVIC

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
UNIVERSITY OF MASSACHUSETTS
AMHERST, MA 01003

SEMI-ANNUAL REPORT FOR PERIOD 15 DEC 82 to 15 JUNE 83.

DISTRIBUTION STATEMENT

Approved for public release;
distribution unlimited.

PREPARD FOR
CENTER FOR COMMUNICATIONS SYSTEM (C. GRAFF)

CECOM

U S ARMY COMMUNICATIONS-ELECTRONICS COMMAND
FORT MONMOUTH, NEW JERSEY 07703

CLEARANCE OF TECHNICAL REPORTS FOR PUBLIC RELEASE

(AR 360-5, AR 70-31 and AR 530-1) SUBMIT IN DUPLICATE

TO: ERADCOM Tech Spt Actv DELSD-L-S	FROM: DRSEL-COM-RF-2 CENCOMS	DATE: 27 July 1983

INSTRUCTIONS: SECTION I: Self explanatory. SECTION IIa: Self explanatory; b, c, d, & e: If the report contains any of this information, Distribution Statement B applies (see DOD Dir 5200.20; f & g: Delete this information from report; h,i & j: Coordinate and obtain approval from agencies concerned. SECTION III: Self explanatory. SECTION IV: Self explanatory.

In compliance with AR 360-5, AR 70-31 and AR 530-1 the attached unclassified technical report is forwarded for clearance for public release and assignment of Distribution Statement A: "Approved for public release; distribution unlimited."

SECTION I DESCRIPTION

1. TITLE OF REPORT
Decentralized Controll of Scheduling in Distributed Processing Systems

2. AUTHOR(S) EXT NO
Dr. John A. Stankovic

FOR CONTRACT REPORTS

3. CONTRACTING OFFICER'S TECHNICAL REPRESENTATIVE COTR EXT NO
CHARLES J. GRAFF 65619

4. CONTRACTOR Dept. of Electrical & Computer Engineering, University of Massachusetts, Amherst, Massachusetts 01003

5. CONTRACT NO DAAB07-82-K-J015 REPORT NO Semi-Annual

6. CONTRACTOR ☐ IS ☒ IS NOT AUTHORIZED ACCESS TO CLASSIFIED MATERIAL UNDER THIS CONTRACT

SECTION II BASIS FOR RELEASE

The report identified above has been reviewed by the undersigned who affirm that the document does not contain any of the following information:

a. Classified information.
b. Information furnished by a foreign government with the understanding that it not be released outside the U. S. Government.
c. Proprietary information for which authority for release has not been obtained.
d. Information resulting from test or evaluation of commercial products or military hardware.
e. Management reviews, records of contract performance evaluation, or other advisory documents evaluating programs of contractors.
f. Information that would be prejudicial or embarrassing to the Government, a foreign government, a contractor, a corporation or an individual.
g. Information concerning subjects of potential controversy among the military services.
h. Subject matter concerning significant policy within the purview of other agencies.
i. Subject matter that implies official positions or scientific attitudes of higher authority.

j. Subject matter on space, satellite, atomic or CW/BR activities requiring clearance of agencies concerned.

SECTION III OPSEC CLEARANCE

In addition, the undersigned certify that the attached R&D technical report has been cleared for public release in accordance with the OPSEC Plan of this activity as prescribed by AR 530-1 and DARCOM supplement 1 thereto.

John E. Quigley
JOHN E. QUIGLEY
OPSEC OFFICER

Charles J. Graff
CHARLES J. GRAFF
AUTHOR (OR COTR)

John E. Quigley
for JAMES H. SALTON
DIVISION DIRECTOR

SECTION IV CLEARANCE

TO:	FROM: ERADCOM TECH LIBRARY, TSA STINFO DELSD-L-S	DATE:

The report identified in Section I is cleared for public release. It may be made available to the National Technical Information Service. It may also be presented at symposia or published in the open literature in the United States without further clearance, provided no change is made contravening the basis of release certified to in Sections II and III.

CHIEF, STINFO OFFICE

SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A131548	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Decentralized Control of Scheduling in Distributed Processing Systems		5. TYPE OF REPORT & PERIOD COVERED Semi-Annual 15 Dec 82 - 15 June 83
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Dr. John A. Stankovic		8. CONTRACT OR GRANT NUMBER(s) DAAB07-82-K-J015
9. PERFORMING ORGANIZATION NAME AND ADDRESS Dept. of Electrical & Computer Engineering University of Massachusetts Amherst, Massachusetts 01003		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS CDR US ARMY CECOM DRSEL-COM-RF-2 Fort Monmouth, NJ 07703		12. REPORT DATE 20 June 1983
		13. NUMBER OF PAGES 74
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution unlimited

Cleared for Public Release

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Distributed Control, Scheduling, Distributed Processing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This semi-annual report details the research program made in the area of distributed control of scheduling using Bayesian Decision Theory. Three decision algorithms were simulated and results compared.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

Semi-Annual Report

Professor John A. Stankovic

Dept. of Electrical & Computer Engineering
University of Massachusetts
Amherst, Massachusetts 01003

Contract Number: DAAB07-82-K-J015
Date of Report: June 20, 1983
Title of Report: Decentralized Control of Scheduling in Distributed Processing Systems
Period: December 15, 1982 - June 15, 1983
Report prepared by: John A. Stankovic

Table of Contents

1. Introduction

This report contains a brief summary of results attained in the last six months (section 2). We also itimize our research plans for the next six months (section 3). A budget summary (section 4), references (section 5), a list of reports prepared during this period (section 6) and acknowledgements (section 7) are also contained herein. Finally, a new paper produced as part of this contract is attached as an appendix. This paper describes, in depth, our work with Bayesian Decision Theory. A summary of these results appears in section 2.1.

2. Brief Summary of Results

2.1 Bayesian Decision Theory

Extensive simulation of a decentralized control algorithm for job scheduling based on Bayesian Decision theory has been performed. The main results show that the algorithm can dynamically adapt to the quality of state information being processed. The need for this adaptive capability grows with the decline in the quality of information available. The decline in quality of state information is due to delays and failures in the communication subnet.

Specific observations of our simulations include: that once tuned, the algorithm works in a stable manner over a wide variety of conditions and over a large number of runs, that the probability distributions describing the true states of nature and the likelihoods of an observation (these distributions are part of Bayesian decision theory (see the Appendix for more details)) converge to values representing the level of observability of the sytem, and that the loss of monitor nodes (included to update probability distributions and recalculate maximizing actions of Bayesian decision theory) does not cause major problems. The simulations also show that our algorithm works well under light loads but simpler approaches would be just as good, that the need for our algorithm increases both for moderate loads and, as stated above, for a decrease in the accurracy of state information available, and several thresholds included in the algorithm improve the operation of the algorithm.

These results and several others are reported in depth in the paper found in the Appendix.

2.2 Scheduling with Real-Time Constraints

Tasks in many distributed systems have severe real-time constraints. Examples of such systems are, nuclear power plants and flight control software. These tasks have execution deadlines and may have precedence constraints and specific resource requirements. Most current research on scheduling tasks with real-time constraints is restricted to multiprocessing systems and hence are inappropriate for distributed systems. With today's advances in software, hardware and communication technology for distributed systems, it may be possible to deal with distributed real-time systems in a more flexible manner than in the past. Our efforts are directed at developing distributed task scheduling software for loosely-coupled systems with the goal of achieving reliability and flexibility.

Multiprocessor scheduling in a hard real-time environment has been described by Mok and Dertouzos [MOK78] in terms of a token game. According to their analysis

1. Earliest deadline scheduling in the case of a single processor is optimal
2. In the multiprocessor case, earliest deadline is not optimal.
3. For two or more processors, no scheduling algorithm can be optimal without a priori knowledge of (i) deadlines (ii) computation times, and (iii) state-times of the tasks.
4. In general, optimal scheduling is an NP-hard problem and is hence computationally intractable [GRAH79].

Leinbaugh [LIEN82] has developed analysis algorithms which when given the device and resource requirements of each task and the cost of performing system functions determines an upper bound on the response time of each task. Our intention, on the other hand, is to develop scheduling strategies so that tasks meet their real-time constraints.

Distributed task scheduling

It should be clear that a practical scheduling algorithm has to be based on heuristics in order to reduce scheduling costs and will have to be adaptive. This is the context in which we have been studying the problem of scheduling in distributed systems.

Our research involves scheduling tasks, with real-time constraints, on processors in a network. A task is characterized by its start time, computation time and deadline. We assume that schedulers on each node in the network interact with one another in the process of scheduling new tasks. Tasks may be periodic or non-periodic. From a scheduler's point of view, a periodic task represents tasks with known (future) start times and deadlines, whereas, a non-periodic task may arrive at any time and may have arbitrary deadlines. In both cases, we assume that a task's computation time is known a priori. Tasks may arrive at any node in the network. When a new task arrives, an attempt will be made to execute the task at that node. If this is not possible, then the scheduler on the node interacts with the schedulers on other nodes in order to determine the node on which the task can be sent to be scheduled.

Functioning of the scheduler on a node

Underlying our scheduling algorithm is the notion of guaranteeing a task. A task is said to be guaranteed if under all circumstances it will be scheduled to meet its real-time requirements. Thus, once a task has been guaranteed, all that is known is that the execution of the task will be completed before its deadline; exactly when it will be scheduled is dependent on the scheduling policy, the tasks that have been guaranteed but are waiting to be executed, the nature of periodic tasks, etc.

As mentioned earlier, for a uniprocessor, the earliest deadline algorithm is optimal. In our scheme, guaranteed tasks are executed

according to the earliest-deadline-first scheme. On each node, information about its surplus processing power, i.e., the surplus after allocating processing power for guaranteed tasks, is maintained in a surplus table. This information is used to expedite the guaranteeing of newly arriving tasks. The surplus table is maintained by assuming that tasks are executed at the last possible instance, just sufficient to meet the deadline. This makes it possible to guarantee new tasks with earlier deadlines. Compared to the token game played in [MOK78] in order to schedule new tasks, we believe that the maintenance of surplus decreases the overheads involved in scheduling. It should be mentioned that our use of the earliest deadline algorithm for scheduling tasks on a single node does not guarantee an optimal schedule on the network as a whole, which, as mentioned earlier, is a computationally infeasible problem. Our aim here is to guarantee tasks quickly and to reduce overheads. Our simulation studies are an attempt to analyze the algorithm's behavior.

Interaction between schedulers on multiple nodes

The scheduler on a node interacts with those on other nodes in a network to determine where a new task could be guaranteed. We propose to utilize

1. the knowledge contained in each scheduler regarding the surplus processing power in other nodes, and
2. a bidding approach wherein a node with a task to be guaranteed requests for bids from nodes which may have sufficient surplus [SMIT80].

Details of this scheme are currently being developed. Some of the policy decisions to be made pertain to

1. the information that needs to be exchanged between nodes in order for each to possess knowledge, albeit incomplete and partially correct, about other nodes,
2. the parameters of the new task that needs to be sent to the bidders,

3. the specification of eligibility to be a bidder,
4. the specification of bids,
5. the length of time a node should wait before it processes bids,
6. the determination of the "highest bidder",
7. the inclusion of communication time in the scheduling computations, and
8. the heuristics that can be used to reduce scheduling costs.

Evaluation of the Scheduling Algorithm

Currently we are in the process of analyzing our scheduling algorithm through simulation of the algorithm's behavior in the presence of both periodic and non-periodic tasks. The present simulation, which deals with scheduling tasks on a single node, is being upgraded to include the bidding scheme whereby it will be possible to schedule tasks arriving at any node in a network. This will also permit the study of different scheduling alternatives and their performance and the tradeoffs entailed by them.

2.3 Other Simulation Results

Simulation results of three adaptive, decentralized controlled job scheduling algorithms have been attained [STAN83a]. The results provide insight into the workings and relative effectiveness of the three algorithms, as well as insight into the performance of a special type of decentralized control. Our simulation approach included tuning the parameters of each algorithm, and then comparing the three algorithms based on response time, load balancing and the percentage of job movement. Each of the algorithms is compared under light, moderate, and heavy loads in the system, as well as a function of the traffic in the communication subnet and the scheduling interval. A general observation is that, if tuned correctly, the decentralized algorithms exhibit stable behavior and considerably improve performance (response time and load balancing) at modest cost (percentage of job movement). Overall, the results contribute to the understanding of a speical type of decentralized control algorithm that has not been extensively studied but is becoming more and more important. These algorithms do not work as well as the Bayesian Decision Theory algorithm but are less costly. We now briefly describe each of the algorithms.

Algorithm 1 For moderate loads in the system, each entity compares its own busyness to its observation (estimate) of the busyness of the least busy host. Note that the host thought to be least busy is itself an estimate. The difference between the busyness of these two hosts is then compared to a bias. If the difference is less than the bias, then no job is moved, else, one job is moved to the least busy host. Jobs are not moved to oneself.

Algorithm 2: Each entity compares its own busyness to its observation (estimate) of the busyness of every other host. All differences less than or equal to bias1 imply no jobs are moved to those hosts. If the difference is greater than bias1 but less than bias2 then one job is moved there. If the

difference is greater than or eq' ·l to bias2 then two jobs are moved there. In no case are more than (y) (z) jobs moved at one time from a host, where y = fraction of jobs permitted to be moved, and z = number of jobs currently at this host. If there is more demand for jobs than an entity is permitted to move, it satisfies the demand in a pre-determined fixed order.

Algorithm 3: This algorithm performs in the same way as algorithm 1 except when an entity, e_i, sends a job to host k at time t, it records this fact. Then for time delta t, it records this fact. Then for time delta t, called a window, this entity will not send any more work to host k. If at any time during the window period, entity e_i calculates that host k is least busy then no job is moved during such an activation. Of course, during the window, jobs may be sent to other hosts if they are observed as least busy by greater than the bias (same bias as in algorithm 1).

One prime motivation behind these three algorithms is that they are all very simple and inexpensive to run, necessary conditions for scheduling algorithms. In algorithm 1 the relative busyness between host i and the least busy host (plus a bias) is used to determine if a job should move. This is about the simplest algorithm we can devise. Since algorithm 1 only moves jobs to the least busy host we felt that a better algorithm might be to spread the work around, i.e., move some work to all the lightly loaded hosts from the heavily loaded ones. Hence algorithm 2 was devised. Finally, we were worried that jobs in transit to a lightly loaded host were not taken into account possible producing instabilities. For example, if (1) host 1 was very busy, (2) host 5 was least busy, (3) host 1 were activated every 2 seconds, and (4) it took 16 seconds for jobs to research host 5, then host 1 could conceivably send at least 8 jobs to host 5 before the first one was received. Other hosts could be doing the same thing. This could result in an unstable situation. Algorithm 3 was designed to avoid such problems.

By comparing performance of the three algorithms to each other and to analytical results, it was concluded that while algorithms 1 and 3 worked well algorithm 2 was the best. An important result is that very simple (execute fast) decentralized controlled job scheduling algorithms can effectively improve performance. Another result is that by utilizing a minimal amount of state information (number of jobs) it was shown that one gets better performance than the optimal fractional assignment which we obtained analytically. Another modification that we suggest is to avoid moving very large (in size) jobs because this congests the subnet and degrades overall response time. This additional state information is easy to obtain and the added execution time checks required are trivial.

2.4 Survey Paper

A survey of current research in distributed systems software was prepared [STAN83b]. Emphasis was placed on research in distributed operating systems, programming languages and databases. An attempt was made to categorize current solution techniques for the individual issues. Important open research questions were itemized including:

1. Distribution of Control: Decentralized control algorithms for various functions of operating systems, such as task scheduling and resource allocation, are needed especially those concerned with a high degree of cooperation between decentralized controllers. Investigation of scheduling concepts such as bidding, clustering, co-scheduling, pause time, wave scheduling is required. The use of various mathematical models such as adaptive control, stochastic control, statistical decision theory, and stochastic learning automata for dealing with uncertainty, inaccuracies and delay in distributed systems is also necessary. Scheduling tasks with real time constraints on a loosely coupled distributed system has received little attention to date due to the difficulties involved.

2. Distribution of processes and resources: It is an open question on how to distribute processes that cooperate to execute a given task. This would affect the topology of the resulting network of processes and the manner in which individual nodes are designed. A crucial question is whether movement of processes in execution is worth it and what the best means to implement such movement is. Tradeoffs between static and dynamic allocation of resources should be investigated. Directory assignment, replication and partitioning should also be addressed. For client - server models of distributed file systems where do we divide the responsibility between file servers and clients? When should distributed file systems be embedded in the operating system and when should a file server model be used? How should the operating system itself be distributed?

2.5 Stochastic Learning Automata

We have been working on a heuristic to perform decentralized job scheduling based on Stochastic learning automata. The actual heuristic has been described in previous reports. During this period the complete simulation program for a network of nodes, each operating as a stochastic learning automata, has been implemented and is currently being debugged. We expect to compare results of these simulations with the Bayesian Decision theory approach completed during this period (see Appendix). A possible problem that we might encounter is a very high cost of simulation runs in order to get the automata to converge on the proper set of actions. This may limit the amount of simulation we want to perform.

2.6 Other Activities

In this section we list several additional activities that were done in conjunction with this contract:

1. Presented a paper [STAN83c] at INFOCOM83, in April 1983.

2. Guest speaker at Distributed Artificial Intelligence Workshop held in June 1983 in Holyoke Massachusetts. Presented portion of [STAN83b] and the paper found in the Appendix on Bayesian Decision theory. The purpose of the talk was to discuss issues and results found in distributed operating systems and in decentralized control algorithms that might apply to artificial intelligence systems.

3. Research Plans

Work will continue on debugging, improving, and extending the three simulation programs developed over the last eight months. These programs are scheduling simulations for (a) a general bidding model, (b) an algorithm based on stochastic learning automata, and (c) a single host algorithm that can handle real time constraints (this last algorithm will be extended to a network simulation).

Once we are convinced that the simulation programs are accurate we will begin evaluation runs. Evaluation of the simulation programs results is expected to take a considerable amount of time.

We also plan to investigate estimation techniques that can eventually be added to the above simulation programs.

We expect that research emphasis will shift to the scheduling algorithm based on bidding since this is more sophisticated and takes issues such as clustering, resource requirements, precedence, etc. into account. The simpler algorithms evaluated assume no a priori information and therefore do not address these issues.

4. Budget Summary

4.1 The Current budget for the 2nd year of this contract is:

Month	Planned Amount	Actually Spent
December (82)	3,857.06	3,857.06
January (83)	2,575.83	2,575.83
February	4,000.00	3,412.64
March	4,000.00	449.30
April	4,000.00	1,834.66
May	4,000.00	1,142.33
June	7,000.00	
July	7,000.00	
August	7,000.00	
September	3,397.64	
October	3,000.00	
November	3,000.00	
December	3,000.00	
TOTAL	55,830.53	13,271.82

4.2 Cummulative Cost to date:

1. 30,236.47 (first year expenditure) + 13,271.82 = 43,508.29

4.3 Revised Budget for remainder of 2nd year of contract

Month	Planned
December (82)	3,857.06
January (83)	2,575.83
February	3,412.64
March	449.30
April	1,834.66
May	1,142.33
June	8,000.00
July	8,000.00
August	8,000.00
September	6,558.71
October	4,000.00
November	4,000.00
December	4,000.00
TOTAL	55,830.53

5. References

[GRAH79] Graham, R. L. et al., "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey", Annals of Discrete Mathematics, 5, 1979, 287-326.

[LEIN82] Leinbaugh, D. W. and Yamini, M., "Guaranteed Response Times in a Distributed Hard-Real-Time Environment", Proc. of the Real-Time Systems Symposium, December 1982

[MOK78] Mok, A. K. and Dertouzos, "Multiprocessor Scheduling in a Hard Real-Time Environment", Proc. of the Seventh Texas Conference on Computing Systems, November 1978.

[SMIT80] Smith, R. G., "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", IEEE Transactions on Computers, C-29, 12, December 1980.

[STAN83a] Stankovic, J. A., "Simulations of Three, Adaptive, Decentralized Controlled, Job Scheduling Algorithms", submitted to Computer Networks, January 1983.

[STAN83b] Stankovic, J. A., K. Ramaritham and W. Kohler, "Current Research and Critical Issues in Distributed Software Systems", submitted to ACM Computing Surveys, April 1983.

[STAN83c] Stankovic, J. A., "A Heuristic for Cooperation Among Decentralized Controllers", Proceedings INFOCOM83, April 1983.

6. Papers Prepared or Presented During this Report Period

Stankovic, J. A., "Simulations of Three, Adaptive Decentralized Controlled, Job Scheduling Algorithms", submitted to Computer Networks, January 1983

Stankovic, J. A., K. Ramaritham and W. Kohler, "Current Research and Critical Issues in Distributed Software Systems", submitted to ACM Computing Surveys, April 1983.

Stankovic, J. A., "A Heuristic for Cooperation Among Decentralized Controllers", Proceedings INFOCOM83, April 1983.

Stankovic, J. A., "Bayesian Decision Theory and Its Application to Decentralized Control of Job Scheduling," to be submitted to IEEE Transactions on Computers.

7. Acknowledgements

I would like to thank Professor Krithi Ramamritham for his cooperation in our work on scheduling with real-time constraints and acknowledge his writing of section 2.2.

8. Appendix

8.1 Attached is a paper entitled "Bayesian Decision Theory and Its Application to Decentralized Control of Job Scheduling".

February 1983

Bayesian Decision Theory and Its Application to Decentralized Control of Job Scheduling

by

John A. Stankovic

Department of Electrical and Computer Engineering

University of Massachusetts

Amherst, Massachusetts 01003

Keywords: Decentralized Control, Job Scheduling, Statistical Decision Theory, Distributed Processing, Bayesian Decision Theory, Simulation, Cooperation, Heuristic Technique

Abstract

There is a wide spectrum of techniques that can be aptly named decentralized control. However, certain functions in distributed operating systems, e.g., scheduling, operate under such demanding requirements that no known optimal control solutions exist. It has been shown elsewhere that heuristics are necessary. This paper presents a heuristic for the effective cooperation of multiple decentralized components of a job scheduling function. An especially useful feature of the heuristic is that it can dynamically adapt to the quality of the state information being processed. Extensive simulation results that show the utility of this heuristic are presented. The simulation results are compared to several analytical models and a baseline simulation model. The job scheduling heuristic presented here is based on an extension of Bayesian decision theory. Bayesian decision theory was used because (a) its principles can be applied as a systematic approach to complex decision making under conditions of imperfect knowledge, and (b) it can run relatively cheaply in real-time.

1.0 Introduction

Depending on the application and requirements, decentralized control can take on many various forms [1] [16] [17] [21] [22] [28] [33] [44]. Although research has been active for all types of decentralized control, the majority of the work is based on extensions to centralized solutions where the entire state is known and can be more accurately described as decomposition techniques, rather than decentralized control. In such work, large scale problems are partitioned into smaller problems, each smaller problem being solved, for example, by mathematical programming techniques, and the separate solutions being combined via interaction variables. In many cases the interaction variables are considered negligible and in others they are limited in the sense that they model very limited cooperation. See [21] for an excellent summary of these types of decentralized control (decomposition). In many of these cases it is an irrelevant detail that individual subproblems are solved in parallel on different computers.

A number of surveys relating to decentralized control have also appeared [17] [31] [38]. These survyes note the unclear meaning of optimality for decentralized control and hypothesize the need for a completely different approach. One such approach is based on the concept of a domule which a combination of a decision agent and its subsystem model [44] [45]. Using this concept, interesting heuristics are proposed for the class of decentralized control problems where significant cooperation is required, but these heuristics are based on decomposition techniques. We are interested in addressing another form of decentralized control where cooperation among the decentralized

controllers is significant as in the domule model but where decomposition is not possible to any large degree. The job scheduling function of distributed operating systems can be implemented with this type of decentralized control.

As a point of comparison, consider two forms of decentralized control: decentralized control that arises in distributed databases, and decentralized control that arises for stochastic replicated functions, such as job scheduling. By replicated functions we mean that the decentralized entities (controllers) implementing a function are involved in the entire problem, not just a subset of it. While the approach of replicating the function may be inefficient for large scale problems, it is appropriate for <u>certain</u> functions like job scheduling.

Solutions to the decentralized control problem in distributed databases are known [1] [2] [27]. The integrity of the database must be maintained by decentralized controllers in the presence of concurrent users. The decentralized controllers must somehow cooperate to achieve a system-wide objective of good performance subject to the data integrity constraint. The cooperation is achieved in various ways, e.g., by the combined principles of atomic actions and unique timestamps, or by the combined principles of two-phase locking and atomic actions, or by certification techniques. It can then be proven that multiple decentralized controllers can operate concurrently and still meet the data integrity constraint [2]. The data integrity constraint simplifies the problem and makes it solvable, but at the same time it lowers performance since users must sometimes wait. However, for other functions such as job scheduling, there is no requirement for absolute data integrity. In fact, if such a constraint were required for scheduling, then, in general, performance of the system would suffer dramatically. Removing the data integrity constraint from the scheduling algorithm improves its performance, but the control problem becomes much more difficult. In fact, in the most general form, solutions to decentralized control of stochastic replicated functions such

as job scheduling are not known [22]. In this paper a heuristic for this type of decentralized control is presented.

Section 2 describes the characteristics of the decentralized control problem for job scheduling as well as some of the related scheduling research. Section 3 presents our heuristic for decentralized job scheduling based on an extension to Bayesian decision theory. The heuristic is adaptive and addresses stability, the assignment of credit problem, the execution cost of running the heuristic itself, the noisy environment and the goal of system-wide performance. Section 4 describes the simulation and analytical models used in this research. Section 5 presents the simulation results for the heuristic as well as comparisons to a baseline simulation model and several analytical models. The overall results show the effective operation of our heuristic. Specific tests illustrate the operation of the heuristic as a function of several parameters of the heuristic, arrival rates, subnet delays, scheduling intervals, and the complete loss of special monitor nodes whose purpose is described later. Section 6 summarizes the conclusions of this study.

2.0 Background and Characteristics of the Problem

Most of the research on scheduling for distributed systems can be considered task assignment research and can be loosely classified as either graph theoretic [3] [6] [41] [42] [43], queueing theoretic [5] [20], based on mathematical programming [7] [8] [24], or heuristic [4] [10] [11] [34]. In most of these cases a task is considered composed of multiple modules and the goal is to find an optimal (or in some cases a suboptimal) assignment policy for the modules of an individual task. Typical assumptions found in task assignment work are: processing costs are known for each module of the task, the interprocess communication costs (IPC) between every pair of modules is known, IPC is considered negligible for modules on the same host, and reassignment is not performed.

Scheduling for extremely large distributed systems, e.g., Wave Scheduling [47], and attempting to cluster related jobs on a host [19] [29] are forms of scheduling receiving more attention. These are not treated in this paper because they require a priori knowledge about arriving jobs.

Other distributed scheduling research is based on the bidding scheme [13] [33]. Here, specific tasks are matched to processors based on the current ability of the processors to perform this work. These schemes are suboptimal and require a priori knowledge, but are more extensible and adaptable than many of the other approaches. However, the cost of making and acquiring bids may become excessive, and the factors to use in making the bids have not been extensively studied.

Our approach is an alternative to the above methods that does not require any priori knowledge, is highly decentralized and adaptive, and operates probabilistically.

We wish to consider a very demanding set of requirements. Consider the job scheduling function to be part of a distributed processing operating system

[9] [12] [18] [35]. The scheduling function itself is to be dynamic, decentralized, adaptive, asynchronous, and operate in a noisy, error prone and uncertain (stochastic) environment. An important aspect of the environment is that significant delays in the movement of data (jobs and state information) are commonplace. Furthermore, the delayed effects of the interactions (decisions) make it difficult if not impossible to know the direct system-wide effect of a particular action taken by a controller. For example, assume that controller i takes action a_i at time t and assume that the net effect of all the actions of all the controllers at time t improve the system. It cannot be assumed that action a_i was a good action, where, in fact, it may have been a bad action dominated by the good actions of other controllers. This problem is sometimes referred to as the assignment of credit problem. One major advantage of our heuristic (as we shall see) is that it is able to deal with this problem in a cost effective manner.

It is important to note that the stochastic nature of the system being controlled affects two distinct aspects of Job Scheduling: an individual controller's view of the system is an estimate, and future random forces can effect the system independently of the control decision. Throughout this paper the job scheduling function is considered as implemented by n decentralized replicated entities (controllers). For reliability we require that there is no master controller - in other words each of the entities is considered equal (democratic) at all times. Furthermore, one of the most demanding requirements is that the Job Scheduling function must run in real-time with minimum overhead (time-sensitive). This requirement eliminates many potential solutions based on mathematical or dynamic programming.

Central to the development of a decentralized Job Scheduling function is the notion of what constitutes optimal control. However, such a notion for

dynamic, democratic, decentralized, replicated, adaptive, stochastic, and time-sensitive (D^3RAST) functions has not yet been well formulated. In fact, this is such a demanding set of requirements that we have not found any mathematical techniques that are directly applicable [38]. However, it seems that Bayesian decision theory can be extended to provide a heuristic for this control problem.

More precisely, our approach for scheduling in distributed systems is to divide the scheduling function into two parts: a decentralized job scheduler (DJS) and a decentralized process scheduler (DPS). Incoming jobs are placed on the wait queues at their site of entry and the DJS (composed of multiple entities) attempts to keep the number of waiting jobs at each site roughly equal in order to improve response time. Again we emphasize that the DJS has no knowledge about the characteristics of incoming jobs. The DJS, operating in a highly decentralized fashion, does however, maintain state information about the network in order to continually adapt to the state of the network. Reassignment of jobs is possible and may cause looping or other forms of multiple moves. Certain aspects of the DJS are similar to routing research [25] [26] [32], e.g., the concept of passing state information at low cost, but differ in that there are no intermediate sites and our heuristic adapts in a different manner based on Bayesian decision theory. The overhead of the DJS is kept low by running relatively infrequently, by adding cheap monitor nodes, (described in Section 3) and by using an inexpensive heuristic. As an example, included as part of the heuristic is the logic that when the system is observed as heavily or lightly loaded the DJS turns itself off (except for the portion that would recognize movement out of these states). In summary, the intent of the DJS is to keep the amount of waiting work at each site roughly equal in an attempt to improve response time, and to do this with low execution-time cost.

The DPS decides which jobs are activated from the wait queues of the DJS. The DPS deals with multiple modules of a job, their resource requirements, clustering concerns and assignment during execution of the job (process). In both the DJS and the DPS, jobs and the modules, respectively, may move any number of times up to some pre-defined limit. This paper describes the results obtained by using Bayesian decision theory (BDT) as a basis for the DJS. The DPS is not discussed any further.

The motivations for splitting the scheduling function into two parts, are:

1) The DJS can be simple and therefore, it is possible to study the use of BDT on simpler decentralized control situations, (in this work we are attempting to understand how multiple, interacting decentralized components of a single function interact), and
2) to eventually study how 2 (or more) decentralized control algorithms interact with each other, e.g., the interaction of the DJS and DPS may provide insight into how an entire system composed of multiple decentralized control algorithms might function.

3.0 Bayesian Decision Theory

The general model for Bayesian decision theory [15] [23] [46] contains five ingredients and a set of maximizing actions. The ingredients are:

1) The set of available actions, $A=\{a_1, a_2, \ldots\}$.

2) The set of the states of nature which can occur $\Theta=\{\theta_1, \theta_2, \ldots,\}$ and the probability distribution on the states of nature, $P(\Theta)$.

3) The utility function $\mu(\Theta, A)$, which contains the consequences of each combination of action and state of nature.

4) A set of possible observations $Z=\{z_1, z_2, \ldots,\}$ and the likelihood distribution $P(Z|\Theta)$.

5) The choice criterion is to maximize expected utility.

Using the above ingredients a simple set of calculations is performed to produce a set of maximizing actions [15] [30] [40]. The set of maximizing actions is a list of what action to take for each possible observation. Hence, once these calculations are performed, a controller needs only to perform a table look-up to determine what action to take given that it has made a particular observation. See Appendix I for a more detailed description of Bayesian decision theory.

The heuristic for applying this general model to decentralized control algorithms is to model each of the n job scheduling controllers as a Bayesian decision maker with the states of nature and observations being defined network-wide (see Appendix I). Periodically, state information is passed between the n controllers so that each controller has a reasonable approximation about the state of the network. Precisely how state information is passed is described in Section 4 when describing the simulation model. The period of update is an important parameter both for cost and for the relative accuracy of the data involved. In adition, special monitor nodes of the network act to

dynamically adjust the probability distributions $P(\theta)$ and $P_i(Z|\theta)$, $i = 1, 2, \ldots n$ by gathering statistics to recalculate the maximizing actions, and to downline load these maximizing actions to each of the n scheduling controllers. This dynamic re-calculation of maximizing actions is a centralized aspect to our heuristic and can also be considered a form of cooperation because each controller informs a centralized component of its true state and its observed state at time t. This information is then used to alter the individual probability distributions needed by each of the controllers. This centralized aspect of our heuristic is permitted because (a) it is a convenient way to calculate the true state (note that it is a true state that has occurred in the past) and, (b) if the centralized monitor node crashes the decentralized controllers can continue to function using the last or a default set of maximizing actions until a backup monitor is activated.

Coordination between the decision makers is accomplished in an implicit manner by passing state information around the network. In other words, there is no direct feedback to a controller of the system-wide goodness of its particular action which is often difficult if not impossible to obtain. Feedback occurs implicitly through the changing states of nature and the updating of the probability distributions. For example, as state information progresses through the network it affects the observations of the different controllers and thereby affecting their control decisions. The quality of the coordination and the resulting performance of the decentralized control algorithm is measured and tuned by simulation. In practice the tuning would be done on a real system. As a result, the assignment of credit problem is finessed. Of course, many other forms of cooperation are possible (and we are working on some of them) but any based on iterations of communications between controllers or any based on sequential processing (e.g., controller i goes

first, then controller i+1, ...) are inappropriate for job scheduling due to their high cost. One aspect of using Bayesian decision theory needs some further discussion: the a priori probability distributions $P(\theta)$ and $P(Z|\theta)$.

3.1 A Prior Probability Distributions

Each controller maintains a table of state information that contains its own view of the state of the network, i.e., it believes some θ_j is the true state. The controller's view at a particular instant of time is called an observation.

More formally, the conditional probability that host i observes z_1 when the true state of nature is θ_1 is written as $P_i(z_1|\theta_1)$. In general, the values $P_i(z_K|\theta_j)$ vary for each controller and over time. However, with the addition of two microprocessors (monitor nodes) shown in Figure 1 we can deal with the dynamics of the system. One microprocessor serves as a monitor that calculates new probability distributions and maximizing actions as the system changes. The second microprocessor is also available and used only if the first monitor becomes unavailable due to failures. Hardware cost of these microprocessors is negligible. In some cases connection costs to the network for these microprocessors may be significant, but we are primarily interested in local networks where these costs should be small or negligible. By adding the monitor the additional processing for the Bayesian decision theory calculations and the updates to probability distributions are done in parallel with the normal functioning of the system. Message overheads are not significant because the messages are small (as described below) and periodic.

In the development of the proposed heuristic using Bayesian decision theory, the immediate problem is how to obtain $P_i(Z|\theta)$ i = 1, 2,...n. Remember, each host periodically adjusts its view of the network by obtaining state information from its neighbors. After such an update, at time t, host 4's state vector might look like this

	Host No.	No. of Jobs (or some other busy est.)
	1	15
	2	12
On Host 4	3	7
	4	10
	5	20

At this point a small amount of additional processing can convert this information into the observation. In this example the above state information converts to z_4, i.e., conditions are moderate and host 3 is least busy by 1-4 jobs (see Appendix I for definition of Z). A simple message to the monitor would then include

Host. No.	Observation

In order to calculate $P_i(Z|\theta)$ it is also necessary to calculate the true state of nature, θ_j. This can be accomplished by having each host transmit the number of jobs at its site. This local information is completely accurate. For example, host 4 can determine with probability 1, how many jobs are at its site. Hence, the message to the monitor from a host must contain three fields.

Host. No.	Observation	Host's State

In this example the three fields are

4	4	10

Host messages to the monitor need not be completely synchronized but each host should send one message per update cycle (missing messages can also be handled). Then, for each update cycle when all the messages arrive at the central monitor, the true state θ_j can be calculated for time t using the Host Number and Host's State fields of all the host's messages. Time t is a time in the recent past. Also available is each host's observation . Using θ_j and the host's observation the monitor can update the probability distribution $P_i(Z|\theta)$ i = 1, 2, ...,n depending on what is observed versus the true state of nature. This overall process is continued for an appropriate time T where T is a system parameter that varies depending on the dynamics of the system. After time T the new probability distributions $P(\theta)$ and $P_i(Z|\theta)$ i = 1, 2, ...n are used in calculating new maximizing actions for each host and these new maximizing actions are downline loaded. These messages are also small and in our study only 16 integers need to be passed to each host (see Appendix I). In summary, dynamic updates to the probability distributions are not difficult, although choosing a good message interval t and update interval T could be difficult.

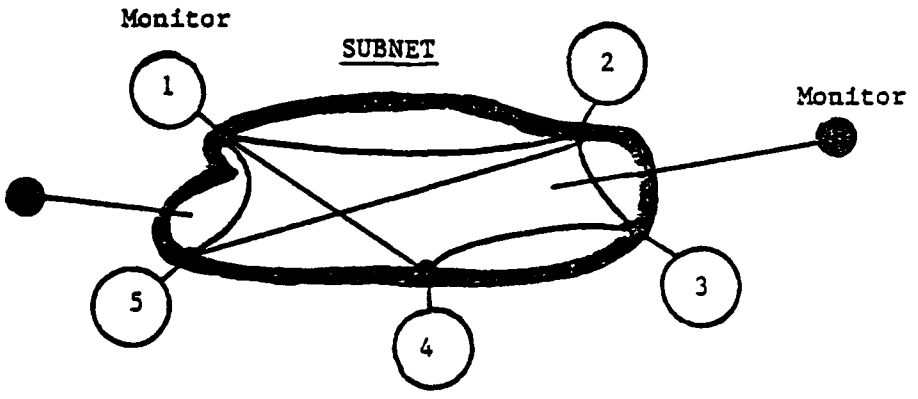


Figure 1: Network Topology

4.0 The Simulation and Analytical Models

The evaluation process for the proposed heuristic includes a baseline simulation model, a Bayesian decision theory (BDT) simulation model and three simple analytical models. Each of these models is described in this section.

4.1 Simulation Models

In our simulations five main characteristics are studied:

1. Parameters of the Algorithm - Several parameters of the BDT model are treated as tunable. These include the biases involved as part of the definition of the state of nature, θ, and the update interval for recalculating maximizing actions.
2. Arrival Rates - Four different sets of arrival rates are used in the simulations (Table 1). The sets of arrival rates are labeled tuning, light, moderate, and heavy. In tuning the baseline and BDT algorithms, it was decided to use a different set of arrival rates than those used for the subsequent algorithm comparisons. This is because, in practice, such algorithms would not be tuned precisely for the current arrival rates. The light and moderate loads are considered the normal network situation. It was decided to simulate a heavy system load to determine how the algorithm performs when, for relatively short times (40 minutes), a system experiences arrival rates greater than its ability to service them.
3. Delay in the subnet - The change in the amount of traffic in the subnet affects response times and load balancing. The affect of this change on the BDT algorithm is studied.
4. Scheduling Interval - The affect of a change in the scheduling interval from 2 -> 4 -> 8 -> 16 seconds is tested.

5. Loss of Monitor Nodes - The BDT heuristic requires updated probabilities provided by monitor nodes. While an enhanced degree of reliability is possible by adding more and more monitors we studied the affect of a complete failure of this updating capability.

While many other characteristics could be studied including size of the network, topology, speeds of the hosts, etc. we felt that the characteristics studied here are some of the most important.

Table 1 - Arrival Rates (Jobs/Second)

HOST	TUNING	LIGHT	MODERATE	HEAVY
1	.18	.1176	.153	.2
2	.1	.1	.125	.2
3	.111	.111	.143	.2
4	.133	.0588	.143	.2
5	.125	.125	.125	.2

The baseline and BDT simulation models are programmed in GPSS and consists of a message based network [37] of five hosts connected as shown in Figure 1. The unit of time in the simulation is milliseconds. Each host is considered identical except for processor speed. The service time of a job scheduled for execution is chosen from an exponential distribution with different averages for each host. The averages are 5000, 7000, 6000, 5000, and 7000 milliseconds for hosts 1-5 respectively. There are five independent sources for arrivals of jobs. Each source is modeled by a Poisson distribution with averages λ_1, λ_2, λ_3, λ_4, λ_5. The λ's vary depending on the particular loads (tuning, light, moderate, heavy) being modeled (Table 1). When a job arrives at a host from the

external world it is assigned a size based on the distribution given in Table 2. Delay in the communications subnet is modeled as a simple function, i.e., the size of the information to be sent divided by the packet size (1K bits) times the average delay per packet. Hence, the delay in the subnet is indpendent of the topology in our simulations. Both jobs and state information are passed into the subnet, thereby modelling two of the major costs involved. Another cost, the cost of running the job scheduling algorithm on each host, is inexpensive since it is primarily a table lookup and passing of state information to neighbors. The Job Scheduling Algorithm is modeled as a fixed cost of 50 milliseconds each time it runs on each host.

Table 2:

Job Size Distribution n (bits):

.1	10,000	.85	22,000
.2	12,000	.9	30,000
.4	14,000	.95	34,000
.6	16,000	.98	38,000
.7	18,000	.99	44,000
.8	20,000	.995	50,000

In the simulations, each host periodically calculates an estimate of the number of jobs at each host in the network, and sends this information to its nearest neighbors. This state information is updated at each host in the following way. Consider three classes of hosts, myself, my neighbors, and all others. In general, host i can determine precisely the number of jobs in its own queue (accurate local data), and therefore, will believe its own value rather than his neighbors perception of his workload. Since the nearest neighbors are only one hop away, their estimates of their workload, as passed to

host i, will be only slightly out of date and, in general, will be a better estimate than estimates other nodes have of them. Therefore, host i uses the nearest neighbors estimate of themselves. All other views of any remaining hosts in the network are determined by taking an average of the estimates from all the incoming update messages. Each job scheduling controller then has an out-of-date observation of how many jobs exist at each site in the network. In future simulations we intend to study the effect of using more sophisticated state information (not just the number of jobs) to estimate the busyness of a host, and to use asynchronous updates of state information rather than periodic.

The simulation program was designed so that it is easy to plug in different job scheduling algorithms (see [39] for results on other algorithms). The job scheduling controllers are activated periodically and also whenever the process scheduler of a host needs to activate a new job for execution. A simple FCFS process scheduler is included that does not allow multiprogramming.

A practical consideration for job scheduling algorithms comes into play for very lightly loaded and very heavily loaded systems. In both instances it is not beneficial to move jobs, therefore, jobs are not moved by a host if it observes a very lightly or very heavily loaded system. Very lightly loaded is defined as each host has less than 4 jobs. A very heavily loaded system is defined as each host has more than 20 jobs. All other situations are considered moderate loads.

In the baseline simulation model each host compares itself to the least busy host and moves 1 job there if the difference is less than a bias. The baseline simulation model is a full network model where the model assumes that 100% accurate information is available. This is considered a type of 'lower bound'. The goal is to have decentralized control algorithms, where inaccurate information does exist, approach the 'lower bound'.

In the BDT simulation, hosts move jobs if the actions calculated via the BDT scheme indicate a move is to take place. This is based on the definition of Θ and Z (see Appendix I) and the calculation of the maximizing actions. The BDT model, of course, uses inaccurate information.

4.2 Analytical Models

4.2.1 Analytical Model - Lower Bound

Analytical solutions to the complex network situation we are simulating are not known. However, by making simplifying assumptions it is possible to analytically obtain some idea of the lower bound.

Assume that the arrival process is Poisson and that service times are described by an exponential distribution. Assume that the service rate μ_i is order so that $\mu_i \geq \mu_j$ iff $i \leq j$. For our network of 5 hosts, this situation is described by the state transition diagram given in Figure 2.

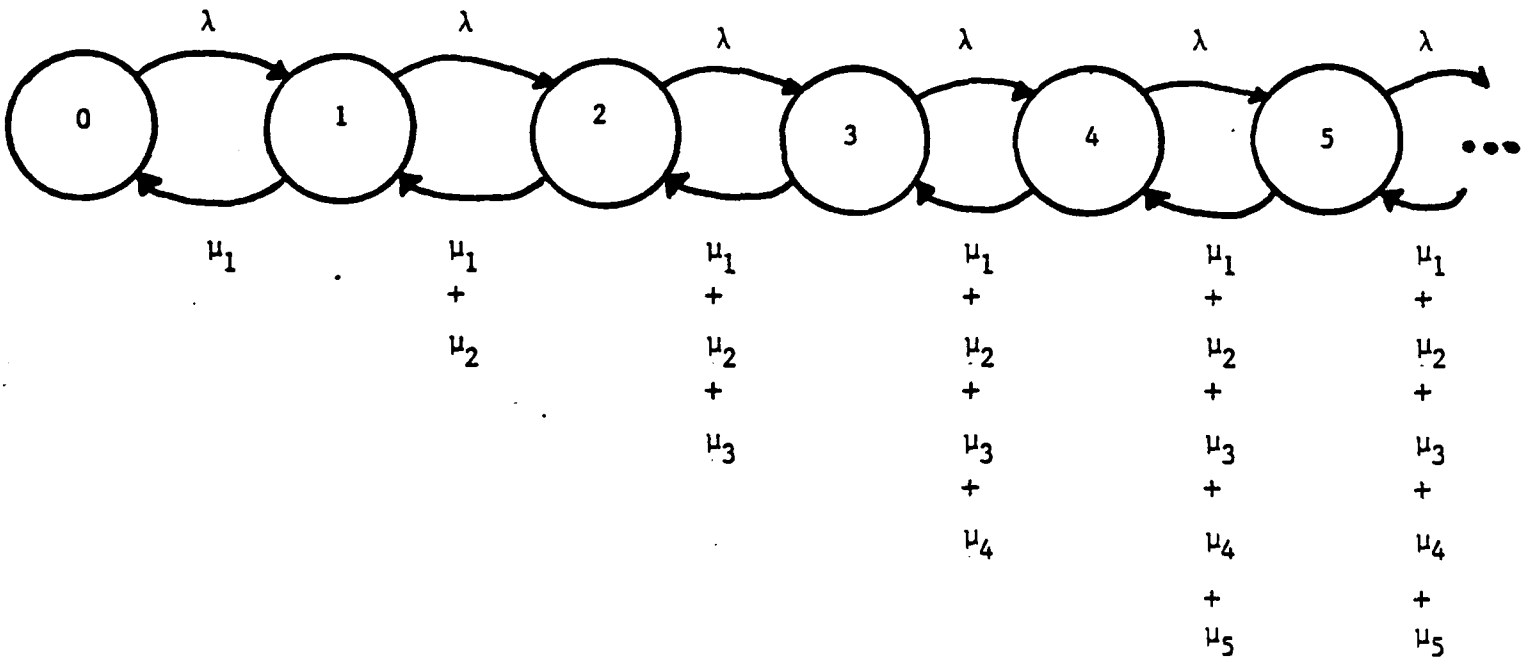


Figure 2 - State Transition Diagram

In general, let the number of processors be N. Define

$$\mu(i) = \sum_{j=1}^{i} \mu_j \quad ,$$

$$M(i) = \prod_{j=1}^{i} \mu(i) \quad i = 1,2,\ldots \quad ,$$

$$M(0) = 1.$$

Then, using the conservation of flow principle and $\sum_j P(j) = 1$, we obtain

$$P_0 = 1/\left(\sum_{j=0}^{N-1} \lambda^j/M(j) + \frac{\lambda^N}{M(N)}/(1-\lambda/\mu(N))\right).$$

The expected queue length is given by

$$\overline{L} = P_0 \left(\sum_{j=1}^{N-1} j \frac{\lambda^j}{M(j)} + \frac{\lambda^N}{M(N)}\left[\frac{N-1}{1-\lambda/\mu(N)} + (1-\lambda/\mu(N))^{-2}\right]\right).$$

Using Little's formula, the expected delay is

$$T = \overline{L}/\lambda.$$

The response times, T, calculated as lower bounds for our given arrival and service rates, are presented in Table 3.

Table 3 - Lower Bound Response Times

System Load	
Light	Moderate
6.16 sec	9.3 sec

This lower bound calculation assumes no communication subnet delay, and that a job can be preempted and moved to a faster processor as soon as that processor becomes available, again at no cost. Although

this analytical model cannot be achieved in practice, it does serve to put a lower bound on our results.

4.2.2 Analytical Model - Fractional Assignment

Assume for a moment that each host knows the fraction of jobs arriving at its site from the external world that it should keep, and the fraction that it should send to each of the other hosts to minimize response time. Such an algorithm would have lower cost than the BDT heuristic because no state information is passed around the network. But how would this fractional assignment algorithm compare as far as response time is concerned? By making some simplifying assumptions, a queueing model can be formed and solved for this fractional assignment algorithm.

Assume a central queue with overall arrival rate $\lambda = \sum \lambda_i$, and 5 hosts with service rates μ_1, μ_2, μ_3, μ_4, μ_5. When jobs enter the central queue they are immediately transferred to one of the hosts based on the optimal fractional assignment, f_i, to each host (Figure 3) to minimize response time.

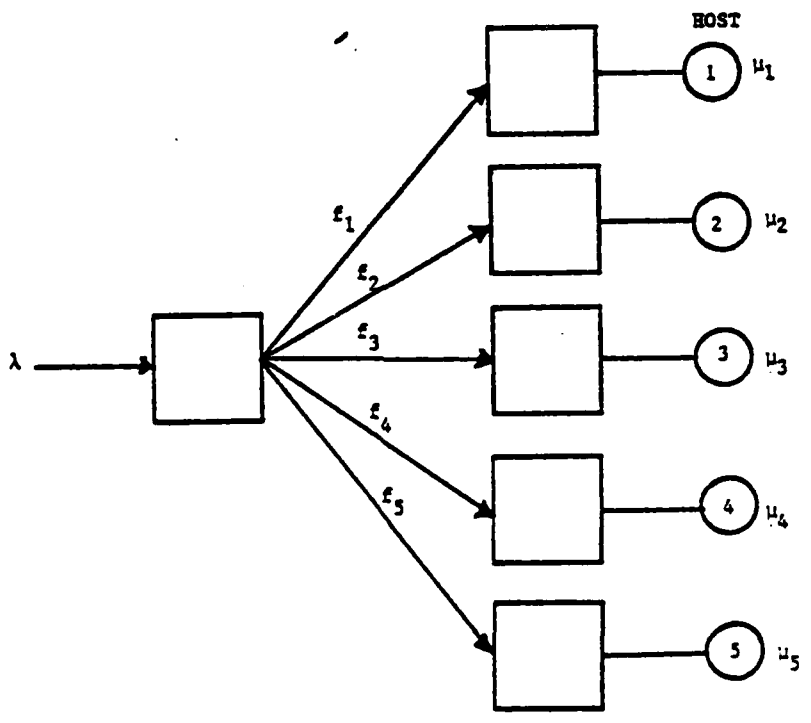


Figure 3: Fractional Assignment Queueing Model

The system response time, T, is given by

$$T = \sum_{i=1}^{5} f_i/(\mu_i - \lambda f_i),$$

and the fraction f_i can be calculated from

$$f_i = \frac{\mu_i}{\lambda} - \frac{(\sqrt{\mu_i})(\sum_{i=1}^{5} \mu_i - \lambda)}{\lambda \sum_{i=1}^{5} \mu_i}.$$

The derivation of f_i is obtained as follows. Starting with

$$T = \sum_{i=1}^{5} f_i/(\mu_i - \lambda f_i),$$

we need to minimize delay, T, subject to $\sum_{i=1}^{5} f_i = 1$. The Lagrangian function can be defined as

$$L = \sum_{i=1}^{5} \frac{f_i}{\mu_i - \lambda f_i} - g(\sum_{i=1}^{5} f_i - 1)$$

where g is the Lagrangian multiplier. Taking the derivative and setting it equal to zero results in

$$\frac{\partial L}{\partial f_i} = \frac{\mu_i}{(\mu_i - \lambda f_i)^2} - g = 0$$

$$f_i = (\mu_i - \frac{\sqrt{\mu_i}}{\sqrt{g}})/\lambda \qquad (1)$$

Since $\sum_{i=1}^{5} f_i = 1$ we have

$$\frac{\sum_{i=1}^{5} \mu_i}{\lambda} - \frac{\sum_{i=1}^{5} \sqrt{\mu_i}}{\lambda\sqrt{g}} = 1$$

$$\sqrt{g} = \sum_{i=1}^{5} \sqrt{\mu_i} \,/\, (\sum_{i=1}^{5} \mu_i - \lambda).$$

Substituting $\sqrt{g}$ into formula (1) gives

$$f_i = \frac{\mu_i}{\lambda} - \frac{(\sqrt{\mu_i})(\sum_{i=1}^{5} \mu_i - \lambda)}{\lambda \sum_{i=1}^{5} \sqrt{\mu_i}}.$$

The results of these calculations for the values used in the simulations are given below in Table 4.

Table 4 - Fractional Assignment Response Times

	System Load	
OPTIMUM FRACTION	LIGHT	MODERATE
.24		
.16		
.20	14.59	30.55
.24		
.16		

Note that the response times calculated here do not contain any delay for actual movement of jobs so this is a very optimistic figure. Further, it was assumed that the arrival rates were known so that the f_i's are the best possible fractions.

4.2.3 Analytical Model - No Network

Here we assume five independent hosts with no network and model each as a simple M/M/1 queue. This serves as an upper bound. The results are average response times of 44.37 seconds for moderate loads and 28.92 seconds for light loads.

The two simulation and three analytical models used for comparison have now been described.

5. Simulation Results

The simulations of the BDT heuristic proceeded in two stages. Stage 1 is static in the sense that no updates of the a priori probabilities and maximizing actions are performed. It is dynamic in the sense of dynamically updating state information and performing job movement. The stage 1 simulations provided evidence that the heuristic could operate effectively and thereby merit further testing. Further, stage 1 simulations model the situation of losing all the monitor nodes, i.e., upon losing the ability to dynamically update the maximizing actions, the BDT scheduling heuristic would switch to maximizing actions modeled in stage 1. The results from stage 1 also serve as a point of comparison to the completely dynamic BDT simulation (stage 2). Stage 1 simulations are run for 10 minutes, statistics cleared and then run for an additional 30 minutes (a total of 2,400,000 time units). This length of run was adequate to reach equilibrium.

Stage 2 is a completely dynamic BDT simulation, dynamically updating the a priori probabilities, maximizing actions, state information and performing job movement. Performing the simulations in this two stage manner provided some interesting results as we shall describe. Stage 2 simulations are run for 10 minutes, statistics cleared and then run for an additional 90 minutes (a total of 6 million time units). As explained later this was adequate to reach equilibrium.

Three measures are used to explain the results of the simulations: system-wide average response time, average number of jobs waiting at each host, and the percentage of job movement. This last term needs some explanation. By percentage of job movement is meant the total number of jobs moved divided by the total number that entered the system. Note that jobs may move more than once and that each move is counted separately because each move adds overhead to the system. This measure is used to provide an indication of whether the

algorithm is stable. Too much movement is considered unstable. The size of jobs is not taken into account in the percentage of movement figure but it is accounted for in response time because jobs are delayed in the subnet proportional to their size. Note that fairness is not treated in this study but it can easily be added simply by keeping track of which jobs move and by limiting their subsequent movement.

5.1 Stage 1: Varying the Bias (Tuning)

Both the baseline simulation model and the BDT model contain the following built in bias: a host i does not move a job to the least busy host if their relative difference in busyness is less than a bias. The purpose of this bias is to avoid moving jobs for small differences in busyness. Without such a bias it is possible to get into unstable conditions of transferring jobs back and forth between hosts. Even with a bias, job looping or a large number of moves of a job are possible, but the probability is lower. This bias is a straightforward addition to the BDT problem description found in Section 3 and Appendix I. A number of simulations were run to choose a reasonable bias for use in subsequent simulations. The results for the BDT model under tuning loads are depicted in Table 5.

For tuning loads, which were chosen as an approximate mixture of light and moderate arrival rates but biased slightly towards the moderate arrival rates, a bias=2 provided only a small loss in response time (14.7$\pm$ .37 seconds) as compared to bias=0 response time but with substantially lower job movement 27.6$\pm$ 1.45 percent as compared to 75 $\pm$ 28.5 percent (Table 5). Bias=2 also proved to be a good choice with light and moderate arrival rates (results not shown - see [36]). A bias=3 resulted in too great a loss in response time (16.5$\pm$.7 seconds). Consequently, for all subsequent simulations the bias was set equal to 2.

Note that for a bias=0 the variation in the percentage of job movement could be considered unstable as shown by the 90% confidence interval (75±28.5%). Percentage of movement greater than 100% is possible because jobs can move more than once. This implies that with the wrong bias this algorithm may not be very stable and in some systems it may be necessary to dynamically adapt this bias. On the other hand, the bias=2 choice proved to be robust over a wide range of arrival rates (tuning, light, moderately, heavy).

Load balancing was very similar for all biases (although the average queue lengths increase with increasing bias) and therefore did not really contribute to the choice of a bias.

Similar results (not shown here) were obtained for the baseline simulation model and therefore we also chose bias=2 for it.

Table 5 - Stage 1: BDT Tuning

	BIASES			
	BIAS=0	BIAS=1	BIAS=2	BIAS=3
Response Time	14.13±.15	14.54±.42	14.7±.37	16.54±.7
% of Movement	75±28.5	47.3±10.7	27.6±1.45	24.3±2.5
Load Balancing (av over 3 runs)				
Host 1	1.06	1.37	1.67	1.80
Host 2	1.28	1.06	1.05	1.50
Host 3	0.86	1.05	0.96	1.34
Host 4	0.85	0.77	0.82	1.31
Host 5	0.82	1.30	1.12	1.29

5.2 Stage 1: Varying the Arrival Rates

Table 6 depicts the 90% confidence intervals for response time and percentage of job movement for various arrival rates. Also shown are average queue lengths.

Under heavy arrival rates very little movement takes place (1.4 ± .6 percent) because of the heavy load threshold in the scheduling algorithm. That is, once a heavy load condition is detected no jobs are moved unless the condition subsides. In this situation the heavy condition results in an unbalanced system, but it does not matter since each host has more than enough work to handle.

The results for the light and moderate arrival rates are of more interest but can be better discussed later (section 5.5) by comparing them to the analytical and baseline simulation models.

Table 6: BDT Scheduling Algorithm

Stage 1: Varying the Arrival Rates

	Light	Moderate	Heavy
Response Time (sec.) (90% Confidence Interval)	14.65±.583	22.18±1.84	Grows Infinitely because arrival rates are faster than service rates
% Of Movement (90% Confidence Interval)	15±1.6	55.2±10.6	1.4±.6
Load Balancing (av. Queue Lengths over 5 runs)			
Host 1	0.74	1.6	33.1
Host 2	0.98	1.65	91.6
Host 3	0.72	1.67	72.1
Host 4	0.21	1.40	46.1
Host 5	1.12	1.53	84.3

Scheduling Interval - 2 seconds

Subnet Delay = 1/2 sec per packet

Bias = 2

5.3 Stage 1: Vary the Busyness of the Subnet

In these simulations the effect of various delays in the subnet (4, 8, 16, and 24 seconds per job) was studied (Table 7). As expected, increased delays in the subnet caused an increase in response time both for light and moderate loads.

Under light loads a job movement delay of 24 seconds reduced the amount of movement considerably (to 8.2±1.7 percent). This is because jobs put into the subnet are in transit for a long time and this makes the system appear more lightly loaded than it is. The correct action under very light loads is to do nothing, hence the reduced movement.

Under moderate loads, response time degrades considerably as delays in the subnet grow but job movement continues to increase, e.g., from 35±2.6 percent with 4 second delay to 91±11 percent with 24 second delay. This is due to two factors. One, jobs in the subnet do not alter the load at the receiving host for quite some time. During this time, scheduling decisions are continuing to be made and even more jobs are being sent to this perceived least busy host. For moderate loads, in contrast to the light loads, there are enough jobs at the various hosts so that jobs are continued to be moved. A possible solution to this problem is to create a window Δt in which a host will not transmit to a site more than 1 job within this window. The window could adapt to the delays in the subnet. Simulation studies using this approach are reported in [39]. Another possibility is to use some form of estimation techniques to predict what other hosts will do.

The second reason for reduced response times under moderate loads is a reduced level of accuracy about the state of the network. State information update messages are also experiencing the increased delays. Since update messages are only one packet in length, their delay varies from 250 -> 500 -> 1,000 -> 1,500 milliseconds corresponding to the 4, 8, 16 and 24 second delays per job. Recall that each job, whose size is taken from a distribution given in

Table 2 is a collection of packets. This decrease in accuracy of state information causes additional movement under moderate loads and contributes to the increase in the degradation of response time as delays in the subnet increase.

Table 7: Stage 1

Varying Subnet Delays

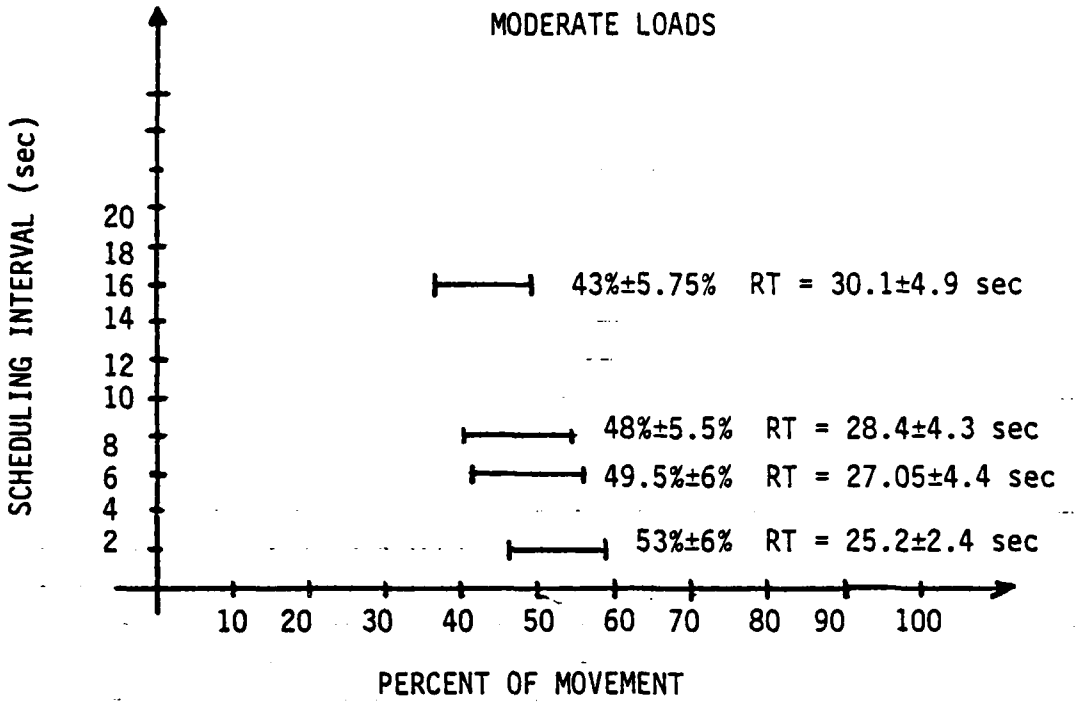
	Average Subnet Delays Per Job			
	4	8	16	25
Light Load				
Response Time	12.7±3	14.65±.583	15.04±.7	16.4±.63
% of Movement	10±2	15±1.6	15.3±.9	8.2±1.7
Moderate Load				
Response Time	17.1±1.3	22.18±1.84	27.7±1	42.04±4.6
% of Movement	35±2.6	55.2±10.6	62.6±4.1	91±11.0

5.4 Stage 1: Vary the Scheduling Interval

For all previous simulation results the scheduler ran at least every 2 seconds, more often if there were job completions. We now tested the effect of slowing down the job scheduler (Figure 4 and 5) and slowing down the subnet to a 1 second delay per packet.

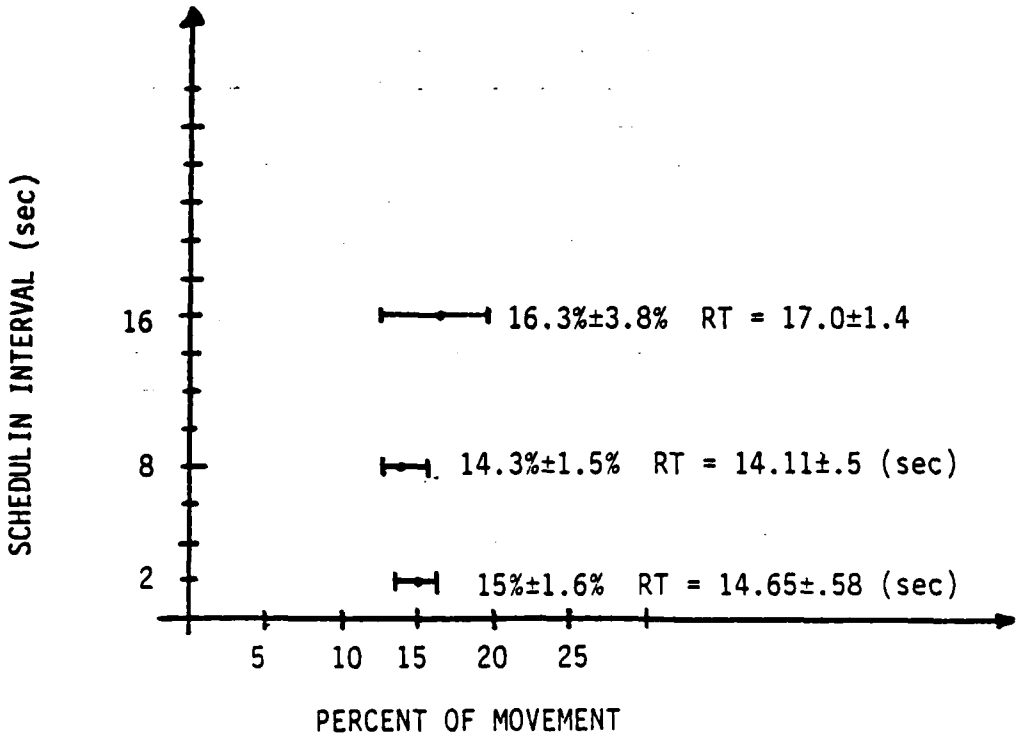
As the interval is lengthened from 2 seconds to 16 seconds there is a steady loss in response time (RT) and a corresponding rise in movement costs (i.e., percentage of movement goes down). If we look at any two sets of data in Figure 4, we can consider this as quantifying the effect of slowing down the scheduler. For example, when the scheduler runs every 2 seconds we have RT = 25.2 $\pm$ 2.4 seconds and percentage of movement 53 $\pm$ 6%; and if the scheduler runs only every 16 seconds we have a response time of 30.1 $\pm$ 4.9 seconds with a 43 $\pm$ 5.75% percentage of movement. Therefore, slowing the scheduling interval from 2 to 16 seconds results in approximately 5 second loss in RT and a gain of about 10% less jobs being moved. Choosing the right scheduling interval is a difficult tradeoff involving not only RT versus percentage of movement but the cost of running the scheduler itself. For moderate loads one might choose 8 seconds as the scheduling interval because it is somewhere in the middle.

FIGURE 4: BDT
STAGE 1: VARY THE SCHEDULING INTERVAL



DELAY IN SUBNET = 1 second per packet

FIGURE 5: BDT
STAGE 1: VARY THE SCHEDULING INTERVAL



DELAY IN SUBNET = 1 second per packet

A different effect is seen under light loads (Figure 5). When the scheduler goes from 2 to 8 seconds there is reduction in movement and a slight improvement in response time. This is due to the fact that under light loads less invocation of the scheduler results in less overhead (these overheads are accounted for in the simulation model), and for scheduling intervals of 2 seconds there is excess movement that is not contributing to improved response time. Finally, using a scheduling interval of 16 seconds results in a loss of response time and an increased movement. This is a result of the scheduler not being invoked often enough to transmit jobs to idle hosts. So, according to this test, we would only have to run the scheduler every 8 seconds for light loads.

5.5 Summary Statistics for Stage 1

Tables 8 and 9 summarize the simulation statistics for light and moderate loads, respectively. These tables also include a comparison to the analytical and baseline simulation models.

Under light loads (Table 8) there is a 50% improvement in response time of the Bayesian decision theory heuristic over the no network case. Also it seems that BDT performs similar to the fractional assignment algorithm. Remember, though, that the fractional assignment response time (14.6 seconds) is very optimistic and does not account for any costs. Hence, using knowledge about the state of the network, our BDT heuristic performs better than knowing the statistical distributions of loads which is necessary to calculate the optimal fractional assignments (see Section 4.2.2). The analytical lower bound for response time of 6.26 seconds is much too low because of the simplifications made in deriving it, and the baseline simulation is a better lower bound (11.76 $\pm$ .3 seconds). BDT then performs within 25% of the baseline which uses perfect information. This difference can be considered as the cost of inaccurate information.

For moderate loads (Table 9) there is again a 50% improvement in response time over for the BDT heuristic over the no network case. Here, though, BDT does considerably better than the fractional assignment (22.18 $\pm$ 1.84 seconds as compared to the optimistic figure of 30.55 seconds for the fractional assignment approach). This is a clear indication that passing reasonably current state information provides improved performance. However, inaccurate information extracts a fairly heavy toll in the moderate load situation by reducing response time from 14.64 seconds (baseline) to 22.18 $\pm$ 1.84 seconds. This seems to be due to too much movement (26% versus 55.2%). This implies that there is room for improvement in the BDT heuristic. The dynamic updates of BDT probabilities and maximizing actions provide some of that improvement (section 5.6, 5.7, 5.8).

Table 8

Stage 1: Summary Statistics - Light Load

	Analytical Models				
	No Network (Upper bound)	Lower Bound	Fractional Assignment	Baseline (av-over 3 runs)	BDT(from Table 6
Response Time (sec)	29.22	6.16	14.6	11.7$\pm$.3	14.65$\pm$.583
% of Job Movement	—	—	—	8$\pm$.7	15$\pm$1.6
Load Balancing					
Host 1	—	—	—	.77	0.74
Host 2				.97	0.98
Host 3				.74	0.72
Host 4				.23	0.21
Host 5				1.09	1.12

Table 9

Stage 1: Summary Statistics - Moderate Load

	Analytical Models				
	No Network (Upper bound)	Lower Bound	Fractional Assignment	Baseline (av-over 3 runs)	BDT(from Table 6)
Response Time (sec)	44.67	9.3	30.55	14.64±2.0	22.18+1.84
% of Job Movement	—	—	—	26±5.1	55.2±10.6
Load Balancing					
Host 1	—	—	—	1.65	1.6
Host 2				1.03	1.65
Host 3				0.96	1.67
Host 4				0.82	1.40
Host 5				1.12	1.53

5.6 Stage 2: Dynamic Bayesian Decision Theory Simulation

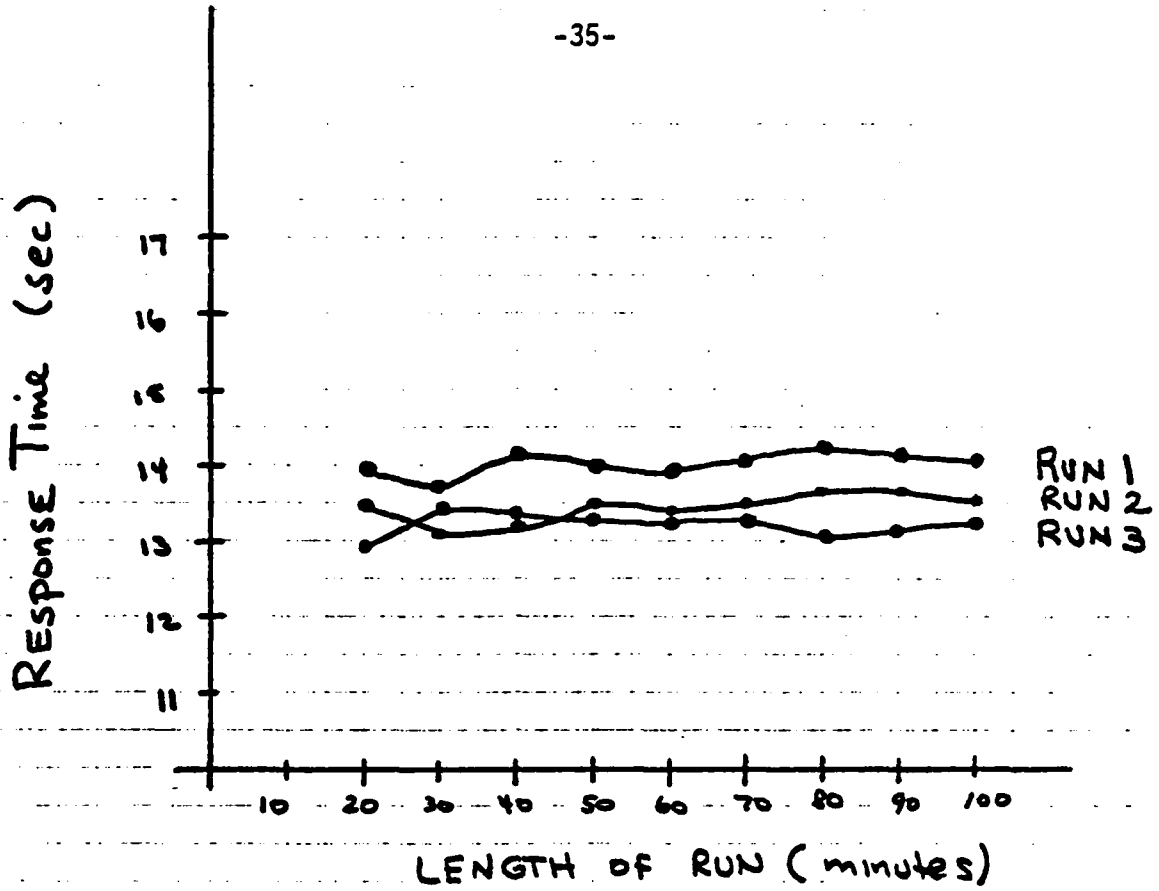
The dynamic Bayesian decision theory simulation has a number of important parameters including (i) the scheduling interval, (ii) the bias, (iii) the period of state information update, (iv) the period of recalculating the maximizing actions, (v) the arrival rates, and (vi) the delays in the subnet. Testing a large range of values for each parameter and in combination with each other is prohibitive. To reduce the problem we chose a scheduling interval of 8 seconds and bias = 2 as derived from the Stage 1 simulation. Then, tuning the DBDT simulation consisted of finding good values for the period of state information update and the period of recalculating the maximizing actions. Finally, we then performed a significant number of tests to determine the effect of various arrival rates (section 5.7) and various delays in the subnet (section 5.8).

The tuning proceeded somewhat subjectively by arbitrarily choosing various values for the period of state information update and for the period of

recalculating the maximizing actions. When the two parameters in combination produced results in line with Stage 1 and the analytical models we assumed that we had reasonable values for these parameters. The result was that the period of state information update needed to be 2 seconds and the period of recalculating the maximizing actions was every 6 seconds. These were held constant in the remainder of the simulations.

5.7 Stage 2: Vary Arrival Rates

In the DBDT tests (Stage 2), the length of the simulation runs are very important because the probability distributions are being determined on line. For light loads the response times and percentage of movement statistics converge almost immediately. The results of three different runs (each run 100 minutes) are shown in Figure 6. For moderate loads it takes about 70 minutes for the probability distributions to converge to the level of observability of the system. After this time, the response times and percentage of movement statistics are fairly constant. Figure 7 shows these results for 3 different runs at moderate loads.



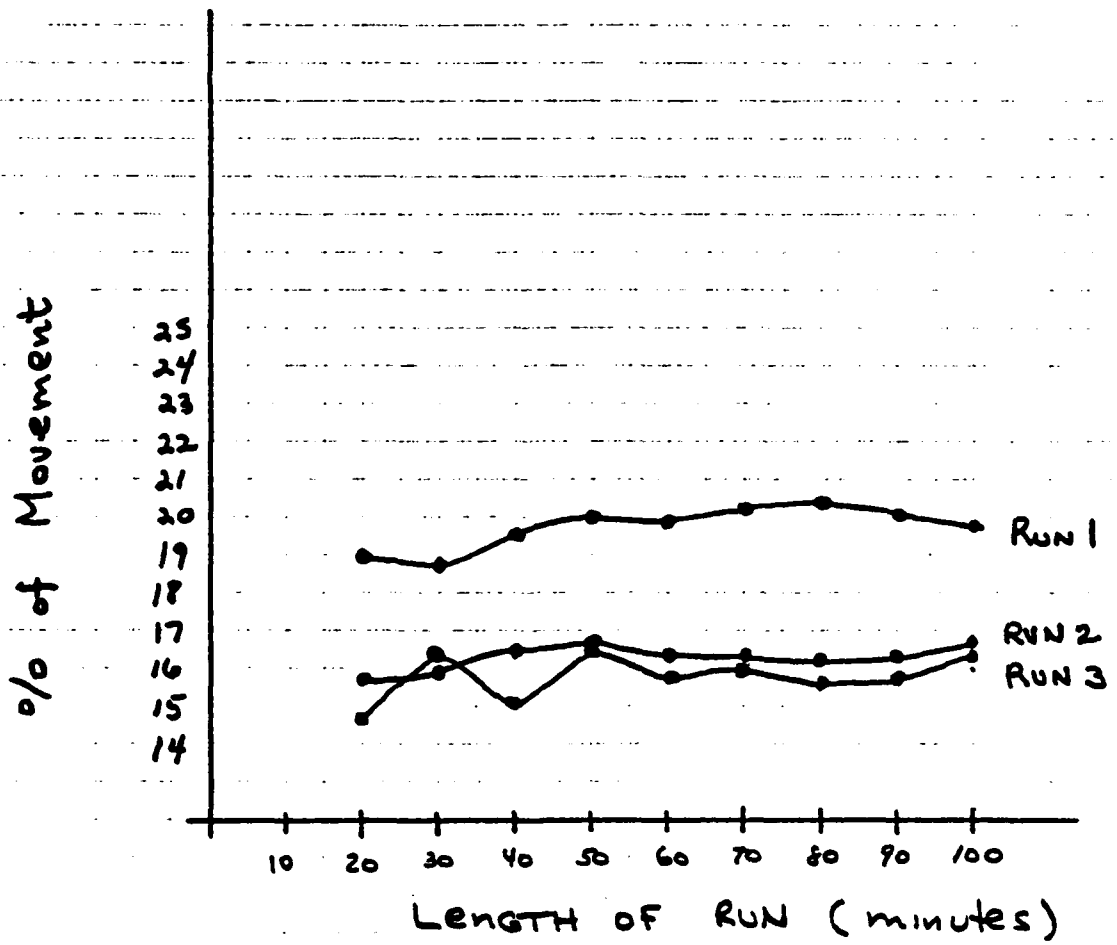
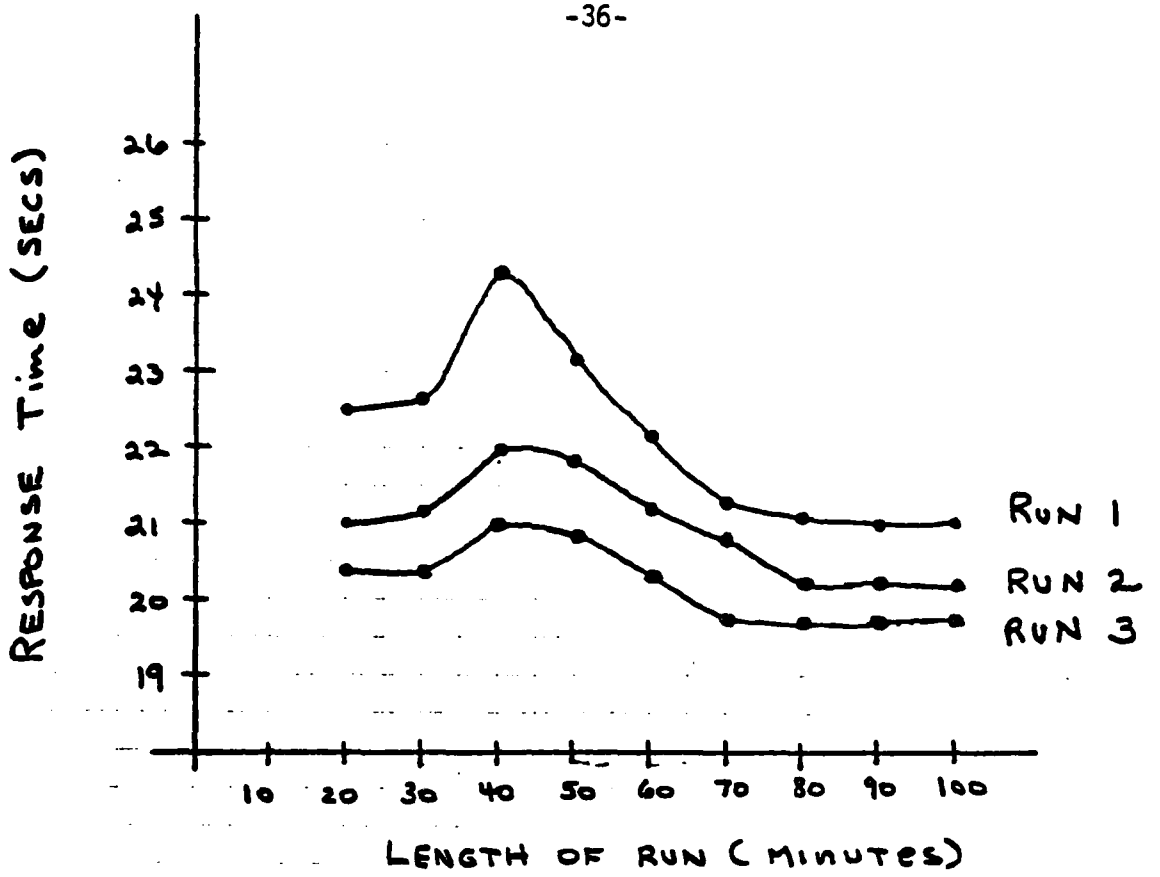


FIGURE 6: DBDT LIGHT LOAD



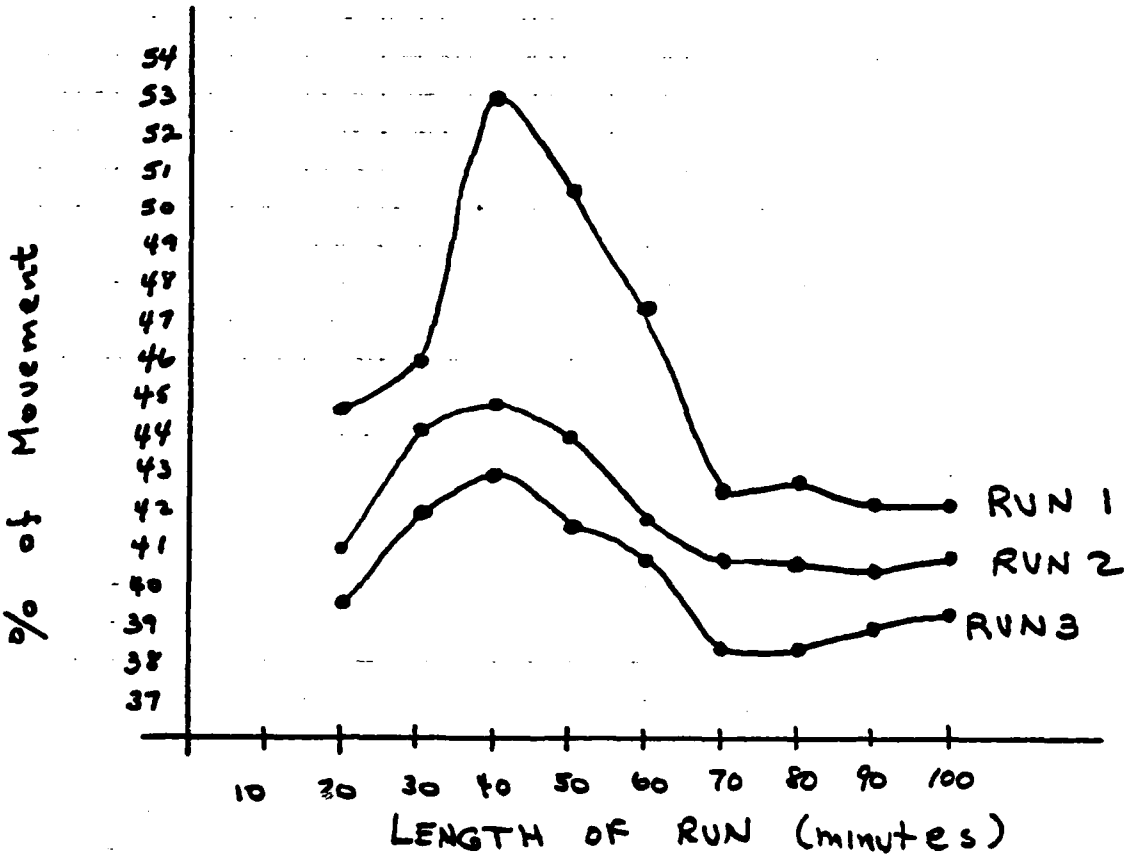


FIGURE 7 : DBDT Moderate Load

Table 10 summarizes the statistics for light and moderate loads using 90% confidence intervals for response times and percentage of job movement after 100 minutes of simulated time. Note that the DBDT response time for the moderate case is 20.06±.8 seconds and the percentage of movement is 40.8±1.5%. For the static BDT simulations (Table 9) the response time was 22.18±1.84 seconds with 55.2+10.6 percentage of movement. The DBDT tests results in both an improvement in response time and a lowering of movement costs due to the dynamic updates of maximizing actions.

Table 10

Stage 2: Light and Moderate Arrival Rates for DBDT

	Arrival Rates	
	Light	Moderate
Response Time (sec)	13.6±.41	20.06±.8
% of Job Movement	18.2±1.1	40.8±1.5
Load Balancing (av-over 4 runs)		
Host 1	0.80	1.78
Host 2	0.74	1.49
Host 3	0.59	1.51
Host 4	0.22	1.33
Host 5	0.70	1.22

Delay in Subnet = 1/2 sec/packet

5.8 Stage 2: Vary Delay in Subnet - and A Comparison to Static BDT Simulations

For the DBDT heuristic under light loads, varying the delay in the subnet (Table 11) results in about the same performance as in the static BDT simulations (compare Table 7 and Table 11). The same arguments used to explain Table 7

apply here. This suggests that under light conditions it is not necessary to pay the cost of updating the maximizing actions.

However, for moderate loads (compare Table 7 with Table 12) the improvement seen in the DBDT heuristic grows as the delay in the subnet grows. The reason is that the DBDT simulation is equipped to identify the fact that the quality of its state information is degrading and uses that in making its decision. For example, summarizing information from Table 7 and Table 12 shows:

Delay per Job	Static RT	Dynamic RT	% Improvement in Response Time
8	22.18	20.06	9.5%
16	27.7	24.8	10.4%
24	42.04	33.4	20.55%

If one also compares the amount of movement (Tables 7 and 12), it is seen that it is significantly reduced from 91% to 54.5% in the case of 24 second delay in the subnet. This kind of performance improvement of the DBDT case over the BDT (static) case is also expected when the network gets larger because there is again reduced accuracy of state information, but it is arising due to separation of hosts and not necessarily to increased traffic in the subnet. In either case the end results are similar.

Table 11

Stage 2: Vary Delay in Subnet-DBDT Light Load

	Delay in Subnet (sec)			
	4	8	16	24
Response Time (sec)	13.4±.3	13.6±.41	16.1±.2	17.2±.6
% of Movement	20.1±1.2	18.2±1.1	21.3±.7	19.5±1.1
Load Balancing (av-over 3 runs)				
Host 1	0.91	0.80	1.0	0.85
Host 2	0.77	0.74	0.78	0.77
Host 3	0.62	0.59	0.77	0.64
Host 4	0.17	0.22	0.17	0.17
Host 5	0.82	0.70	0.80	0.73

Table 12

Stage 2: Vary Delay in Subnet-DBDT Moderate Load

	Delay in Subnet (sec)			
	4	8	16	24
Response Time (sec)	19.9±.6	20.06+.8	24.8±1.1	33.4±1.7
% of Movement	41±2.1	40.8±1.5	45.5±2.1	54.5±3.2
Load Balancing (av-over 3 runs)				
Host 1	2.13	1.78	2.12	2.6
Host 2	1.73	1.49	1.81	2.19
Host 3	1.68	1.51	1.53	1.87
Host 4	1.57	1.33	1.15	1.81
Host 5	1.39	1.22	1.44	1.4

Consider the following, assume that the DBDT heuristic is running and operating in steady state producing an average response time of 20.06 seconds for an 8 second delay in the subnet (Table 10). At this point there is a failure of both monitor nodes. The system simply shifts to the static maximizing actions

which would approximately produce an average response time of 22.18 seconds (Table 7), a fairly small degradation. However, if the delay in the subnet was 24 seconds then the system would be experiencing an average of 33.4 second response time before the failure and an average of 42.04 seconds (Table 7) after the failure. Since this later response time is approaching the no network response time (44.67 seconds), it may be wise to simply turn off load balancing when the average delay per job reaches X, where X would be approximately 24 seconds delay per job in this case. Remember, for a heavily loaded system (all hosts have more than 20 jobs) we are already turning off load balancing, and now we are suggesting adding the requirement to turn off load balancing when the state information becomes too old.

Finally, since the DBDT simulation operates as well as or better than the static BDT simulations (this is seen by the above arguments as well as by further comparing Tables 8 and 9 with Tables 10, 11 and 12), then the comparison to the baseline and analytical models described in Section 5.6 remains valid, further stressing the relatively good performance of our heuristic.

6.0 Conclusion

The form of decentralized control under investigation here has not been dealt with to any large degree because of its intractable nature, although it is becoming increasingly important for distributed computer systems. We have developed a low cost heuristic for decentralized control of job scheduling when no a priori information is known about jobs by extending Bayesian decision theory and studied its performance. Using extensive simulations, our heuristic is shown to provide good response time and load balancing in comparison to analytical and baseline simulation models. The amount of movement necessary to achieve this performance is quantified. A major reason for our heuristic performing well is that it is able to adapt to the quality of the state information it is receiving and use that in making its decisions. This ability becomes increasingly

important for busy or noisy subnets and we hypothesize also for larger and larger networks.

Several specific observations of our simulations include: that once tuned the algorithm works in a stable manner over a wide variety of conditions and over a large number of runs, that the probability distributions describing the true states of nature and the likelihoods of an observation converge to values representing the level of observability of the system, that the loss of monitor nodes does not cause major problems, that the heuristic works well under light loads but simpler approaches would be just as good, that the need for the heuristic increased both for moderate loads and for a decrease in the accuracy of state information available, and several thresholds included in the heuristic improve the operation of the algorithm.

As a final suggestion we believe that the heuristic should be modified to avoid moving very large jobs. For example, when our heuristic decides to move a job, it removes a job off the back of the local queue without regard for size. A better approach would be to first check if this job is very large, if so, move some other job. This should further improve response time.

Appendix I
Bayesian Decision Theory Problem Formulation
Stage 1

(1) Let $A = \{a_j\}$, $j = 0,\ldots 5$:

a_0: move no jobs

a_1: move one job from queue i to host 1

a_2: move one job from queue i to host 2

a_3: move one job from queue i to host 3

a_4: move one job from queue i to host 4

a_5: move one job from queue i to host 5

(2) The states of nature are defined as $\theta = \{\theta_0, \theta_1,\ldots\theta_{16}\}$ where:

θ_0 = conditions are light (i.e., no host has more than 4 jobs), or conditions are moderate (not heavy and not light) and no host is least busy).

θ_1 = conditions are heavy, i.e., all hosts have more than 20 jobs.

Note, all other cases are considered moderate loads. States $\theta_2 - \theta_{16}$ inclusive deal with moderate loads plus the added requirement listed below.

θ_2 - host 1 is least busy and is less busy than host i by 1-4 jobs.

θ_3 - host 2 is least busy and is less busy than host i by 1-4 jobs.

θ_4 - host 3 is leasy busy and is less busy than host i by 1-4 jobs.

θ_5 - host 4 is least busy and is less busy than host i by 1-4 jobs.

θ_6 - host 5 is least busy and is less busy than host i by 1-4 jobs.

θ_7 - host 1 is least busy and is less busy than host i by 5-8 jobs.

θ_8 - host 2 is least busy and is less busy than host i by 5-8 jobs.

θ_9 - host 3 is least busy and is less busy than host i by 5-8 jobs.

θ_{10} - host 4 is least busy and is less busy than host i by 5-8 jobs.

θ_{11} - host 5 is least busy and is less busy than host i by 5-8 jobs.

θ_{12} - host 1 is least busy and is less busy than host i by $>$ 8 jobs.

θ_{13} - host 2 is least busy and is less busy than host i by $>$ 8 jobs.

θ_{14} - host 3 is least busy and is less busy than host i by $>$ 8 jobs.

θ_{15} - host 4 is least busy and is less busy than host i by $>$ 8 jobs.

θ_{16} - host 5 is least busy and is less busy than host i by $>$ 8 jobs.

(3) The Utility Function

Simply stated, the theory of utility makes it possible to measure the relative value to a design team (or an individual) of the payoffs or consequences in a decision problem. This is based on two axioms of utility.

1) If payoff R_1 is preferred to payoff R_2 then $U(R_1) > U(R_2)$; if R_2 is preferred to R_1 then $U(R_2) > U(R_1)$ and if neither is preferred then $U(R_1) = U(R_2)$.

2) If you are indifferent between (a) receiving payoff R_1 for certain and (b) taking a bet or lottery in which you receive R_2 with probability p and payoff R_3 with probability (1-p) then $U(R_1) = pU(R_2) + (1-p)U(R_3)$.

We need to apply these axioms in the assessment of utility functions for job scheduling. In most computer systems analysis, utility is expressed in terms of utilization, throughput or delay. These are all performance related issues. In general, this is not really satisfactory for distributed systems of interest since reliability and performance are equally important requirements and therefore both should be factored into the decision making process although this is usually quite difficult to do. As a simplified example consider the following development of a utility function.

First, the most profitable. R^*, and least profitable R_* payoff must be determined. Then $U(R^*)$ and $U(R_*)$ can be quantified in any way providing $U(R^*) > U(R_*)$. Choose $U(R^*) = 1$ and $U(R_*) = 0$. Next consider any payoff R, then $U(R) \geq U(R_*)$. To determine U(R) more precisely consider the following choice of lotteries.

Lottery I = Receive R for certain.

Lottery II = Receive R^* with probability p
and R_* with probability 1-p.

Then the expected utility of lottery I is U(R) and of lottery II is $pU(R^*) + (1-p)\,U(R_*)$. In general, a decision maker would choose the lottery with higher expected utility. However, if you can determine the probability p that makes you indifferent between the two lotteries, then the utility of R must be equal to p. In this manner you can determine the utility of any payoff, however complicated.

In order to further clarify the determination of a utility function let's consider a simple example for job scheduling. The primary requirements for the job scheduling algorithm are reliability and good performance. Both of these requirements need to be considered when generating the utility function. The worst payoff R_* as a result of a decision would be if that decision resulted in an unreliable situation and poor performance. The best payoff R^* occurs when each of the above requirements is met to the highest degree.

Consider the following situation of a 5 host network. Host 1 and 2 are identical machines but host 1 is more reliable than host 2. Host 3 is the same machine as host 1 and 2 but with limited memory compared to host 1

and 2. It is as reliable as host 1. Host 4 is very fast compared to hosts 1, 2 and 3 but not as reliable as any of them. Host 5 is slow compared to hosts 1-4 inclusive and equally reliable as hosts 1 and 3.

Now, if the state of nature is θ_0 (all hosts are equally busy) and the action is a_0 (no jobs are moved) then no utility is gained or lost from performing action a_0. This serves as a reference point and is assigned the value .5, i.e., $U(\theta_0, a_0) = .5$. Hence values between 0 - .49 indicate losses of utility while values .51 - 1.0 indicate gains of utility.

Now for action a_1 (move a job to host 1) given θ_0 we must have $U(R_*) \leq U(\theta_0, a_1) < U(\theta_0, a_0)$, or $0 < U(\theta_0, a_1) < .5$ because moving a job during conditions of light load is considered a loss of utility. Choose $U(\theta_0, a_1) = .3$. But taking action a_2 (move a job to host 2) given θ_0 is an even greater loss of utility than $U(\theta_0, a_1)$ because host 2 is less reliable than host 1. Hence $0 < U(\theta_0, a_2) < .3$, we choose $U(\theta_0, a_2) = .2$. Taking action a_3 (move a job to host 3) given θ_3 is also a loss of utility. In this case the loss is greater than $U(\theta_0, a_1)$ because host 3 has less memory than host 1 and therefore, in general, jobs may take longer to complete at host 3. On the other hand, host 3 is more reliable than host 2 and we judge that reliability is of more importance than performance so $.2 < U(\theta_0, a_3) < .3$ choose $U(\theta_0, a_3) = .25$. Continuing in this manner designers can assign utilities to each element of the utility function (Table A1). It is generally not critical to choose an exact quantification and a sensitivity analysis can be performed on the utility function [36].

This technique of developing a utility function may, at first, seem to be totally subjective, but it is not. If it is possible to express performance and reliability gains and losses as a mathematical expression, then that expression can be used in developing the utility function. If it is not possible to develop a mathematical expression, then in our view the designers have no other choice but to use the above approach explicitly. Furthermore, too often one is lulled into believing that a mathematical model of a problem is not subjective. This is true only after it is formulated. For example, a discipline such as team theory requires an observation of controller i, Z_i, to be some function of the states of nature, θ, and the controls, U, written as $Z_i = N_i(\theta, U)$ and it requires a loss function, such as,

$$\varepsilon = 1/2(\theta + aU_1 + U_2)^2 + hU_1^2 + qU_2^2 \text{ (for a two member team).}$$

How are the 2 functions N_i and ε choosen? Supposedly they model the true system and are determined by careful thought and analysis. But that is exactly what the methodology for the development of the utility function is.

In summary, even though the development of the utility function is somewhat subjective, attempting to create such a function is a viable methodology that forces designers to consider the factors involved and their interrelationships. This technique is certainly more viable than making decisions in a completely ad hoc manner.

Table A1: The Utility Function

The Utility Function = (θ_i, a_j) is defined

by the following table:

States of Nature θ	A = actions					
	a_0	a_1	a_2	a_3	a_4	a_5
θ_0	.5	.3	.2	.25	.3	.25
θ_1	.5	.3	.2	.25	.3	.25
θ_2	.25	.75	.28	.29	.3	.29
θ_3	.3	.3	.65	.29	.3	.29
θ_4	.3	.3	.28	.7	.3	.29
θ_5	.25	.3	.28	.29	.75	.29
θ_6	.3	.3	.28	.29	.3	.7
θ_7	.15	.9	.23	.24	.25	.24
θ_8	.2	.25	.8	.24	.25	.24
θ_9	.2	.25	.23	.85	.25	.24
θ_{10}	.15	.25	.23	.24	.9	.24
θ_{11}	.2	.25	.23	.24	.25	.85
θ_{12}	0	1.0	.13	.14	.15	.14
θ_{13}	.05	.15	.95	.14	.15	.14
θ_{14}	.05	.15	.13	1.0	.15	.14
θ_{15}	0	.15	.13	.14	1.0	.14
θ_{16}	.05	.15	.13	.14	.15	1.0

(4) The set of possible observations $Z = z_0, z_1, \ldots z_{16}$ where each z_k corresponds to θ_j for $k = j$ except that the z_k's are observations and not true states. The values $P_i(z_k|\theta_j)$ assumed for each host in the static BDT simulations are:

$$P_i(z|\theta)$$

States of Nature	z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}	z_{11}	z_{12}	z_{13}	z_{14}	z_{15}	z_{16}
θ_0	.08	.0035	.0833	.0833	.0833	.0883	.0883	.07	.07	.07	.07	.07	.03	.03	.03	.03	.03
θ_1	.0935	.08	.03	.03	.03	.03	.03	.07	.07	.07	.07	.07	.0833	.0833	.0833	.0833	.0833
θ_2	.04	.003	.14	.08	.08	.08	.08	.1	.0625	.0625	.0625	.0625	.075	.018	.018	.018	.018
θ_3	.04	.003	.08	.04	.08	.08	.08	.0625	.1	.0625	.0625	.0625	.018	.075	.018	.018	.018
θ_4	.04	.003	.08	.08	.14	.08	.08	.0625	.0625	.1	.0625	.0625	.018	.018	.075	.018	.018
θ_5	.04	.003	.08	.08	.08	.14	.08	.0625	.0625	.0625	.1	.0625	.018	.018	.018	.075	.018
θ_6	.04	.003	.08	.08	.08	.08	.14	.0625	.0625	.0625	.0625	.157	.018	.018	.018	.018	.018
θ_7	.045	.045	.07	.07	.07	.07	.07	.24	.03	.03	.03	.03	.15	.0125	.0125	.0125	.0125
θ_8	.005	.005	.11	.11	.07	.07	.07	.03	.24	.03	.03	.03	.0125	.15	.0125	.0125	.0125
θ_9	.005	.005	.07	.07	.15	.07	.07	.03	.03	.243	.03	.03	.0455	.0455	.015	.0455	.0455
θ_{10}	.005	.005	.07	.07	.07	.0375	.07	.03	.03	.03	.24	.03	.0125	.0125	.125	.15	.0125
θ_{11}	.005	.005	.07	.07	.07	.07	.15	.03	.03	.03	.03	.24	.0125	.0125	.0125	.0125	.15
θ_{12}	.0776	.0224	.06	.06	.06	.06	.06	.2	.03	.03	.03	.03	.22	.015	.015	.015	.015
θ_{13}	.0077	.0201	.1183	.1183	.06	.06	.03	.2	.03	.03	.03	.0156	.22	.015	.015	.015	.015
θ_{14}	.0076	.0224	.06	.06	.13	.06	.06	.03	.03	.2	.03	.03	.015	.015	.22	.015	.015
θ_{15}	.005	.0224	.06	.06	.06	.1326	.06	.03	.03	.03	.2	.03	.015	.015	.015	.22	.015
θ_{16}	.005	.0224	.06	.06	.06	.06	.13	.0313	.0313	.03	.03	.2	.015	.015	.015	.015	.22

(5) The Probability distribution $P(\theta)$ assumed for the Static BDT Simulations, is

θ_0	.05	.1	very light at times
θ_1	.05		very heavy at times
θ_2	.1	.5	
θ_3	.1		
θ_4	.1		
θ_5	.1		
θ_6	.1		
θ_7	.04	.2	
θ_8	.04		
θ_9	.04		
θ_{10}	.04		
θ_{11}	.04		
θ_{12}	.04	.2	
θ_{13}	.04		
θ_{14}	.04		
θ_{15}	.04		
θ_{16}	.04		

$$\sum_{i=0}^{16} \theta_i = 1$$

Finally, we show an example of a list of maximizing actions for a particular controller at time t:

z (observations)	a (maximizing action)
z_0	a_1
z_1	a_1
z_2	a_1
z_3	a_1
z_4	a_2
z_5	a_3
z_6	a_4
z_7	a_1
z_8	a_1
z_9	a_3
z_{10}	a_4
z_{11}	a_5
z_{12}	a_1
z_{13}	a_2
z_{14}	a_3
z_{15}	a_4
z_{16}	a_5

References

1. Bernstein, P. A., and N. Goodman, Concurrency Control in Distributed Database Systems, Computing Surveys, Vol. 13, No. 2, pp. 185-221, June 1981.

2. Bernstein, P. A., and D. W. Shipman, The Correctness of Concurrency Control Mechanisms in a System for Distributed Databases (SDD-1), ACM Transactions on Database Systems, 5, 1, pp. 52-68, March 1980.

3. Bokhari, S. H., Dual Processor Scheduling with Dynamic Reassignment, IEEE Transactions on Software Engineering, Vol. SE-5, No. 4, July 1979.

4. Bryant, Raymond M. and R. A. Finkel, A Stable Distributed Scheduling Algorithm, Proc. 2nd International Conference on Distributed Computing Systems, April 1981.

5. Chow, Yuan-Chieh, and Walter Kohler, Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System, IEEE Transactions on Computers, Vol. C-28, No. 5, May 1979.

6. Chow, T. C. K. and J. A. Abraham, Load Balancing in Distributed Systems, IEEE Transactions on Software Engineering, Vol. SE-8, No. 4, July 1982.

7. Chu, W. W., Optimal File Allocation in a Multiple Computing System, IEEE Transactions on Computers, Vol. C-18, pp. 885-889, October 1969.

8. Chu, W. W., L. H. Holloway, M. Lan, and K. Efe, Task Allocation in Distributed Data Processing, Computer, Vol. 13, pp. 57-69, November 1980.

9. Davies, D. W., E. Holler, E. D. Jensen, S. R. Kimbleton, B. W. Lampson, G. LeLann, K. J. Thurber and R. W. Watson, Distributed Systems - Architecture and Implementation, Lecture Notes in Computer Science, Vol. 105, Springer-Verlag, N.Y., 1981.

10. Efe, Kemal, Heuristic Models of Task Assignment Scheduling in Distributed Systems, IEEE Computer, Vol. 15, No. 6, June 1982.

11. El-Dessouki, O. I., and W. H. Huan, Distributed Enumeration on Network Computers, IEEE Transactions on Computers, Vol. C-29, pp. 818-825, September 1980.

12. Enslow, P. What is a Distributed Data Processing System, IEEE Computer, Vol. 11, No. 1, January 1978.

13. Farber, David J., et al., The Distributed Computer System, Proc. 7th Annual IEEE Computer Society International Conference, February 1973.

14. Gonzalez, M. J., Deterministic Processor Scheduling, ACM Computing Surveys, Vol. 9, No. 3, pp. 173-204, September 1977.

15. Halter, A. N. and G. W. Dean, Decisions Under Uncertainty, South-Western Pub. Co., Chicago, IL, 1971.

16. Ho, Y., Team Decsion Theory and Information Structures, Proceedings of the IEEE, Vol. 68, No. 6, June 1980.

17. Jarvis, R. A., Optimization Strategies in Adaptive Control: A Selective Survey, IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-5, No. 1, January 1975.

18. Jensen, E. D., The Honeywell Experimental Distributed Processor-An Overview of its Objectives, Philosophy and Architectural Facilities, IEEE Computer, Vol. 11, No. 1, January 1978.

19. Jones, A. K., et al., StarOS, A Multiprocessor Operating System for the Support of Task Forces, Proc. 7th Symposium on Operating System Principles., pp. 117-127, 1979.

20. Kleinrock, L., and A. Nilsson, On Optimal Scheduling Algorithms for Time-Shared Systems, JACM, Vol. 28, No. 3, pp. 477-486, July 1981.

21. Larsen, R. E,. Tutorial: Distributed Control, IEEE Catalogue, No. EHO 153-7, IEEE Press, NY, 1979.

22. LeLann, G., Distributed Systems-Towards a Formal Approach, Proceedings IFIP Congress, Toronto, North Holland Pub., pp. 155-160, August 1980.

23. Lindgren, B. W., Elements of Decision Theory, The MacMillan Co., NY, 1981.

24. Ma, P. Y. R., E. Y. S. Lee, and M. Tsuchiya, A Task Allocation Model for Distributed Computing Systems, IEEE Transactions on Computers, Vol. C-31, No. 1, pp. 41-47, January 1982.

25. McQuillan, J. M., Adaptive Routing Algorithms for Distributed Computer Networks, BBN Report 2831, May 1974.

26. McQuillan, J. M. and David C. Walden, The ARPS Network Design Decision, Computer Networks, The International Journal of Distributed Informatique, Vol. 1, No. 5, pp. 243-289, August 1977.

27. Menasce, D., G. Popek, and R. Muntz, A Locking Protocol for Resource Coordination in Distributed Databases , ACM Transactions on Database Systems, 5,2 pp. 103-138, June 1980.

28. Narendra, K. S., editor, Proceedings of the Workshop on Applications of Adaptive Control, Yale University, August 1979.

29. Ousterhout, J., D. Scelza, and P. Sindu, Medusa An Experiment in Distributed Operating System Structure, CACM, Vol. 23, No. 2, February 1980.

30. Raiffa, H. and R. Schlaifer, Applied Statistical Decision Theory, Division of Research, Graduate School of Business Adm., Harvard University, Cambridge, MA 1961.

31. Sandell, N., R. Varaiya, M. Athans and M. Safonov, Survey of

Decentralized Control Methods for Large Scale Systems, IEEE Transactions on Automatic Control, Vol. AC-23, No. 2, April 1978.

32. Segall, A., The Modeling of Adaptive Routing in Data-Communication Networks, IEEE Trans. on Communications, Vol. COM-25, No. 1, pp. 85-95, January 1977.

33. Smith, R. G., The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver, IEEE Transactions on Computers, Vol. C-29, No. 12, December 1980.

34. Soloman, M. H., and R. A. Finkel, Roscoe: A Multi-Microcomputer Operating System, Proceedings of the Second Rocky Mountain Symposium on Micro-Computers, pp. 291-310, August 1978.

35. Stankovic, J. A. and A. van Dam, Research Directions in (Cooperative) Distributed Processing, Research Directions in Software Technology, MIT Press, Cambridge, MA 1979.

36. Stankovic, J. A., A Methodology for Making Decisions in Decentralized Control Algorithms, UMASS TR, March 1982.

37. Stankovic, J. A., Software Communication Mechanisms: Procedure Calls Versus Messages, IEEE Computer, Vol. 15, No. 4, April 1982.

38. Stankovic, J. A., N. Chowdhury, R. Mirchandaney, I. Sidhu, An Evaluation of the Applicability of Different Mathematical Approaches to the Analysis of Decentralized Control Algorithms, Proceedings COMPSAC 82, November 1982.

39. Stankovic, J. A. Simulations of Three, Adaptive, Decentralized Controlled, Job Scheduling Algorithms , submitted to Computer Networks, January 1983.

40. Stankovic, J. A. A Heuristic for Cooperation Among Decentralized Controllers, Proceedings INFOCOM 83, April 1983.

41. Stone, H. S. Multiprocessor Scheduling with the Aid of Network Flow Algorithms, IEEE Trans. on Software Engineering, Vol SE-3, January 1977.

42. Stone, H. S. and S. H. Bokhari, Control of Distributed Processes, IEEE Computer, Vol. 11, No. 7 pp. 97-106, July 1978.

44. Tenney, R. R. and N. R. Sandel, Jr., Structures for Distributed Decisionmaking, IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-11, No. 8, pp. 517-527, August 1981.

45. Tenney, R. R. and N. R. Sandell, Jr., Strategies for Distributed Decisionmaking, IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-11, No. 8, pp. 527-538, August 1981.

46. Winkler, R. L., Introduction to Bayesian Inference and Decision, Holt, Rinehard Winston, Inc., NY, 1972.

47. Wittie, L. and Andre M. van Tilborg, MICROS, A Distributed Operating System for Micronet, A Reconfigurable Network Computer, IEEE Transactions on Computers, Vol. C-29, No. 12, December 1980.

Tables and Figures

DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2 copies
Commander USAERADCOM ATTN: DELSD-L-S Fort Monmouth, N.J. 07703	2 copies
Commander USACECOM ATTN: DRSEL-COM-RF (Dr. Klein) Fort Monmouth, N.J. 07703	1 copy
Comander USACECOM ATTN: DRSEL-COM-D (E. Famolari) Fort Monmouth, N.J. 07703	1 copy
Comander USACECOM ATTN: DRSEL-COM-RF-2 (C. Graff) Fort Monmouth, N.J. 07703	10 copies

that it is able to adapt to the quality of the state information it is receiving and use that in making its decisions. This ability becomes increasingly